September 13 , 2013

511 Traffic Program Enhanced Data Fusion System (EDFS) System Architecture and Design

Task Order 6.23

Version 1.8

Prepared by:
Leidos

1000 Broadway, Suite 675, Oakland, CA 94607

Prepared for:
Metropolitan Transportation Commission

101 8th Street, Oakland, CA94607

Revision History				
Date	**Author**	**Notes**	**Reviewer**	**Version**
01/15/2013	Bimesh Giri	First Version for MTC review		1.0
02/21/2013	Bimesh Giri	Updates made based on discussion with Janet, added designs for User, Audit, Log and Utility Modules	David Balmer	1.1
03/04/2013	David Balmer	Added Configuration Module and updated Log and Audit Modules	Bimesh Giri	1.2
03/11/2013	David Balmer	Updated Configuration Module to remove separate configuration message queue requirement	Bimesh Giri	1.3
04/01/2013	David Balmer	Expanded Configuration section.	Bimesh Giri	1.4
04/19/2013	David Balmer Thet Zaw Bimesh Giri	Added Role and User Privileges Added UML sequence diagram for user module Added Data flow diagram showing various queues used by the system	Bimesh Giri	1.5

05/09/2013	David Balmer	Replaced class diagrams for data elements with detailed layouts including field descriptions and lookup tables. Updated to match current development.		1.6
05/24/2013	David Balmer Bimesh Giri Thet Zaw	Updated document as configuration module has to be changed due to issues identified during unit testing Added details on data archiving, alarm processor, publisher and reporting	Bimesh Giri	1.7
09/13/2013	David Balmer Bimesh Giri Thet Zaw	Updated various modules per requirement changes Added section for service failover (active/passive) scenarios. Expanded Alarm module and reports sections	Bimesh Giri	1.8

Contents

1 Acronyms Used

API	Application Programming Interface
AWS	Amazon Web Services
CMS	Content Management System
COT	Congestion Override Tool
CSV	Comma Separated Values
DAL	Data Access Layer
DCOL	Data Collection Subsystem
EC2	Elastic Compute Cloud
EDFS	Enhanced Data Fusion system
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IVR	Interactive Voice Response
JMS	Java Message Service
MLDB	Master Link Database
MSMQ	Microsoft messaging queue
NoSQL	Not Only SQL
OOD	Object-Oriented Design
OOP	Object-Oriented Programming
RDS	Relational Database Service
S3	Simple Storage Service
SES	Simple Email Service
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SQS	Simple Queue Service
TOMS	TravInfo Open Messaging Service
TTL	Time To Live
URL	Uniform Resource Locator
XML	Extensible Markup Language
ZIP	Compressed data format

2 Introduction

2.1 Purpose

This document provides an overview of the system architecture and design of the new EDFS, a web based application that will allow TIC operators to manage incidents/events data within 511. For detailed list of system functionalities, please refer to the requirement specification document. This document describes the data flow and modules that will help construct the new EDFS.

3 Intended Audience

This document describes the new EDFS architecture and design including process flows, modules and components, classes and interfaces between them, data storage and messaging. The intended audience of this document is software developers and engineers, and system administrators who will be involved in the system development and maintenance. This document assumes that reader has knowledge of Object-Oriented Design (OOD), Object-Oriented Programming (OOP), distributed systems, system and software architectural and design patterns, principles of relational and NoSQL databases, SQL (Structured Query Language), XML (Extensible Markup Language), C# and the Java programming languages, XML, web development, Microsoft .NET framework, MSMQ (Microsoft messaging Queues), and Amazon Web Services (AWS). The terms *event* and *incident* have been used interchangeably in this document. Similarly terms such as *state*, *stage* and *situation* refer to same entity called “state” and represent the state of an event.

4 Architecture Principles & Standards

The following architectural principles and standards should guide the design and development of the new EDFS:

Architecture Principles

- Application should have an open architecture to allow easy future extension.
- Application modules should be loosely coupled to allow possible future replacement with other modules.
- Application should support industry standard technologies and best practices.
- Application should be able to reside on a standard Windows Server platform and should not require any non-standard hardware.
- The application should leverage the latest software releases and other technical infrastructure to capitalize on the benefits from emerging industry practices.
- Application architecture should enable business continuity and should be available 24 hours a day, 7 days a week.
- It should be ensured that any failure attributed to the environment or network has minimal impact on application access and the execution of related jobs.
- Recoverability, redundancy and maintainability should be addressed at the time of design.
- Application should be easy to use and maintain.

5 Key Concepts

This section explains key concepts of new EDFS which are useful in understanding the overall system architecture.

5.1 EDFS Data Flow

Event data within EDFS can originate from multiple data sources such as CHP CAD, LCS and EDFS users. To receive data, EDFS exposes an interface to each of these data sources e.g. the CHPDI interface receives data from CHP CAD. Once data is received and processed by the data interface, it will then pass through a set of data processors or enrichers that will add additional information to help construct an Event message that can be injected into EDFS for further processing. Each Data interface can have multiple data processors that it will use to enrich the raw data. These data processors will be available across multiple data interfaces and each data interface can have a configurable data processor workflow that it can enrich its data through.

After data has been enriched, the data interface will then submit the event to the event orchestration engine (EOE) The EOE is the heart of EDFS, and where event will go through it's lifecycle following a configured workflow. The EOE monitors every event in the EDFS that is not in Archived state. Following the workflow, EOE will:

- Transition event states based on defined actions within a workflow.
- Create audit trails for deriving performance metrics.
- Trigger data publishing to external systems such as JMS, API.
- Generate alarms based on certain conditions which then get displayed or sent out to users.

EDFS will also include an Audit and Log component to collect audit and log information.

The EDFS web interface will provide all user interfaces required to manage event, users, workflow and reports. The below diagram depicts this system data flow at a very high conceptual level.

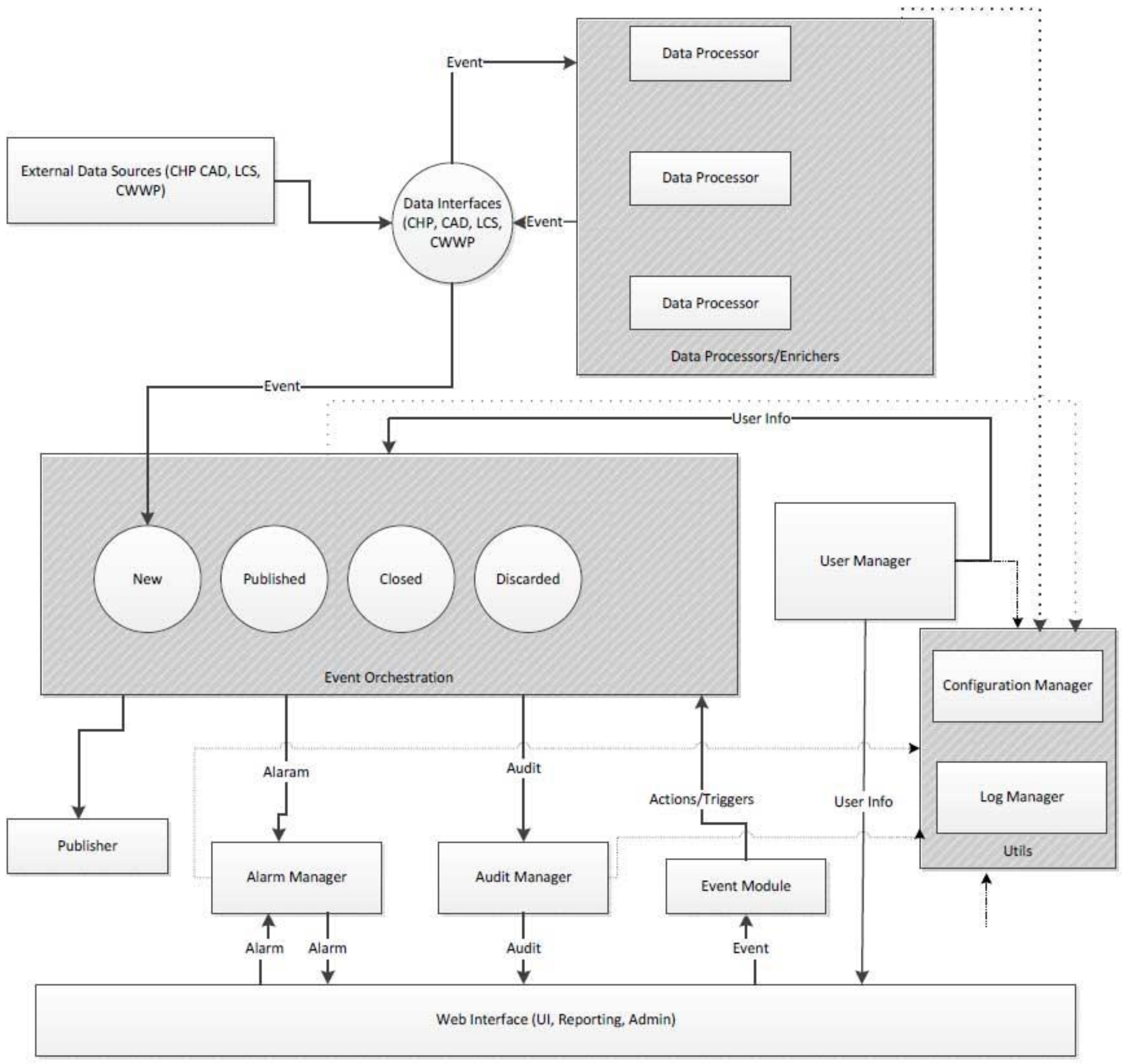


Diagram 1: ESFS Conceptual Architecture

5.2 Event workflow

The primary function performed by EDFS is to manage event workflow, configured by system operators. Events in EDFS will go through a series of states/stages from the time they are created to the time they are archived. States typically represent the "situation" of an event during its lifecycle. The lifecycle of an event will be configurable, based on its source. Each state has set of activities and an event transitions from state to state as a result of execution of one such activity. State transition activity is typically executed by a user; however state transitions can occur autonomously if configured likewise. Some activities just update the current situation of a state without resulting in state transition.

Each state has four activities that are executed as a result of actions by users/system or by the state itself. StateInitialization and StateFinalization activities are executed by virtue of being in a particular state; whereas ChangeState and UpdateState activities are triggered by users or the system.

- **StateInitialization** activity - If this optional activity is defined, it is the first activity the workflow executes when it enters a state. This activity will typically include creating audit logs such as recording the date and time when an event entered this state.

- **ChangeState** activity - This activity will result in state transition. Each state will have a configurable set of other states that it can transition to and users who can initiate such transitions via actions e.g. an incident reported via twitter feed can transition from New state to Published state via the Publish action performed by TIC Supervisors only.

- **UpdateState** activity - This activity will update the current situation of the state without resulting in state transitions. Audit logs and notifications can be generated based on certain conditions when an update occurs.

- **StateFinalization** activity, if this optional activity is defined, it is the last activity the workflow executes as it leaves a state. This activity will typically include creating audit logs such as recording the date and time when an event exited this state. These audit logs can be later used to derive performance metrics e.g. the difference between time logged at StateFinalization and StateInitialization will tell us how long an event stayed in a particular state.

Below are the different states of an event:

- **New** – This represents a situation where an event is automatically created from data received from external system such as CHP CAD, or if an event is entered using the EDFS Web interface. Workflow can be configured such that events can automatically transition to the next state e.g. an event can automatically transition from **New** to **Published** state based on the start time of the event. Similarly configuration can be defined such that specific users will be required to perform the "**Publish**" action for a transition to occur and alerts are sent out to users responsible to take actions. A workflow state cannot have both automatic transitions as well as action driven transitions defined at the same time.

- **Published** – This represents a situation where an event has been published by operator or was created from a trusted system which doesn't require any action from an operator.
- **Closed** – This represents a situation where an event is has been closed.
- **Archived** – This represents a situation where an event has been archived.
- **Discarded** – This represents a situation where an event has been discarded.

EDFS provides set of actions that can be performed on an event in order to change its state or situation. These actions are executed under the ChangeState or UpdateState activity of a state. Listed below are the actions available at each state. Every event type, based on its source will require workflows to be configured using these states and actions. Each action will have list of associated actors (users) who can perform them.

Actions available during "**New**" state are:

- **Publish** – This action involves changing the state of the event to "**Published**" and can be performed either by a user or system. A Publish action configured to be performed by a system will result in automatic transitions.
- **Update** - This action involves updating attributes of an event and can be performed either by a user or system. This action will not result in state change.
- **Discard** – This action involves discarding an event by a user and will result in change of state to "**Discarded**".

Actions that can be performed on an event during "**Published**" state are:

- **Update** - This action involves updating attributes of an event. Depending on the type of user performing this action, will either result in republishing the event with updates or creating a change request requiring a review. The User assigned to approve or apply these change requests will be configurable and alerts will be sent out to all those users whenever a change request is submitted. This action will not result in state change.
- **Close** – This action involves closing an event and will result in change of state to "**Closed**". Events that were published to external systems will be republished as closed.
- **Expire** – This is an internal action that is executed when a scheduled event expires and will result in change of state to "**Closed**".

Actions that can be performed on an event during "**Discard**" state are:

- **Update** - This action involves updating attributes of an event. This action will not result in state change.

- **ReOpen** – This action involves reopening an event by a user. ReOpen will result in transitioning to "**Published**" state.
- **Archive** – This action involves archiving an event and will result in change of state to "**Archived**".

Actions that can be performed on an event during "**Closed**" state are:

- **Update** - This action involves updating attributes of an event. This action will not result in state change.
- **Archive** – This action involves archiving an event and will result in change of state to "**Archived**".
- **ReOpen** – This action involves reopening an event by a user. ReOpen will result in transitioning to "**Published**" state.

Below is a sample data flow for a new **Event** coming into EDFS from an external data source such as CHPCAD.

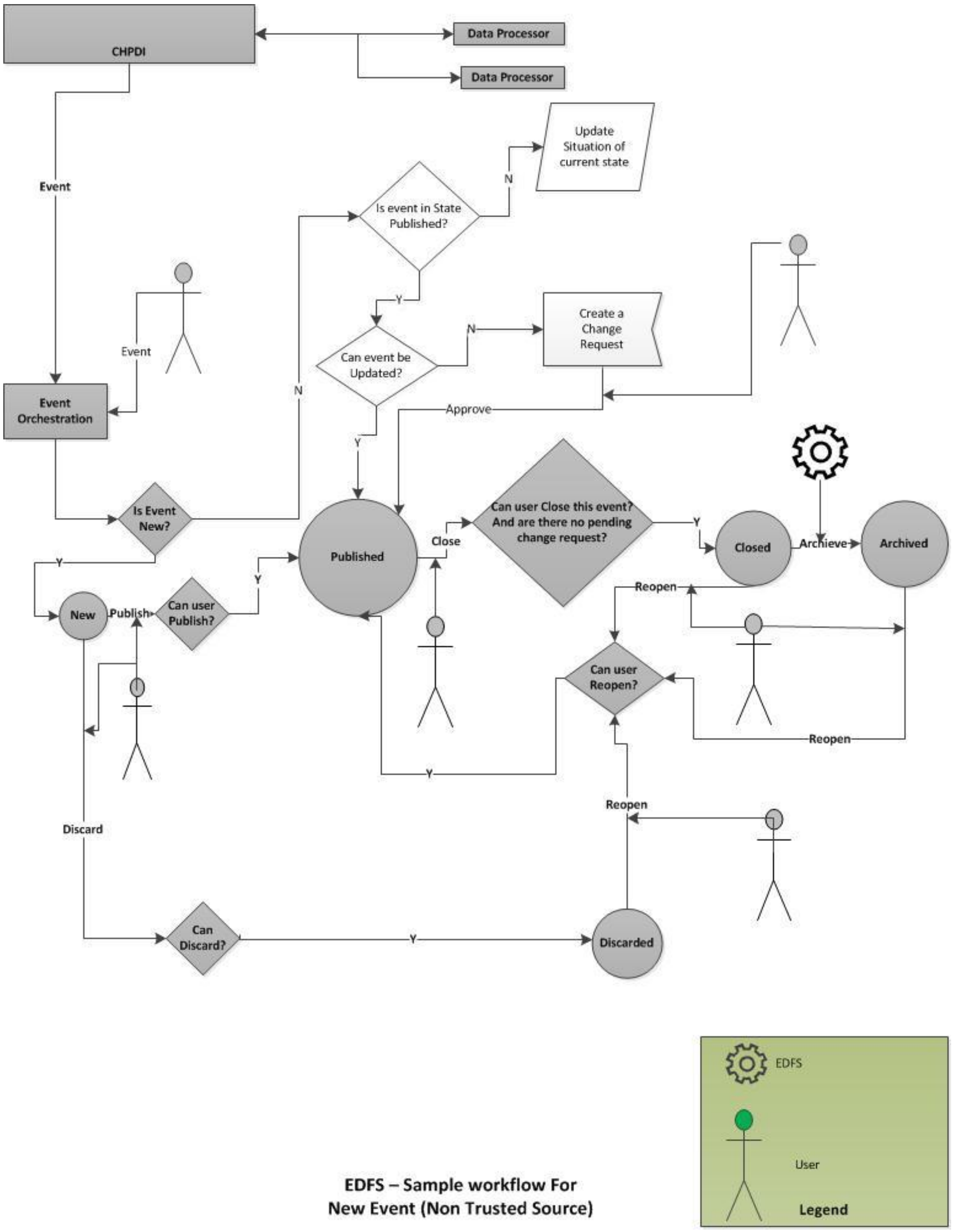


Diagram 2: Sample EDFS Workflow (New Event)

Below is a sample data flow for Event updates coming into EDFS from an external data source or the update being performed by an operator using the EDFS web interface.

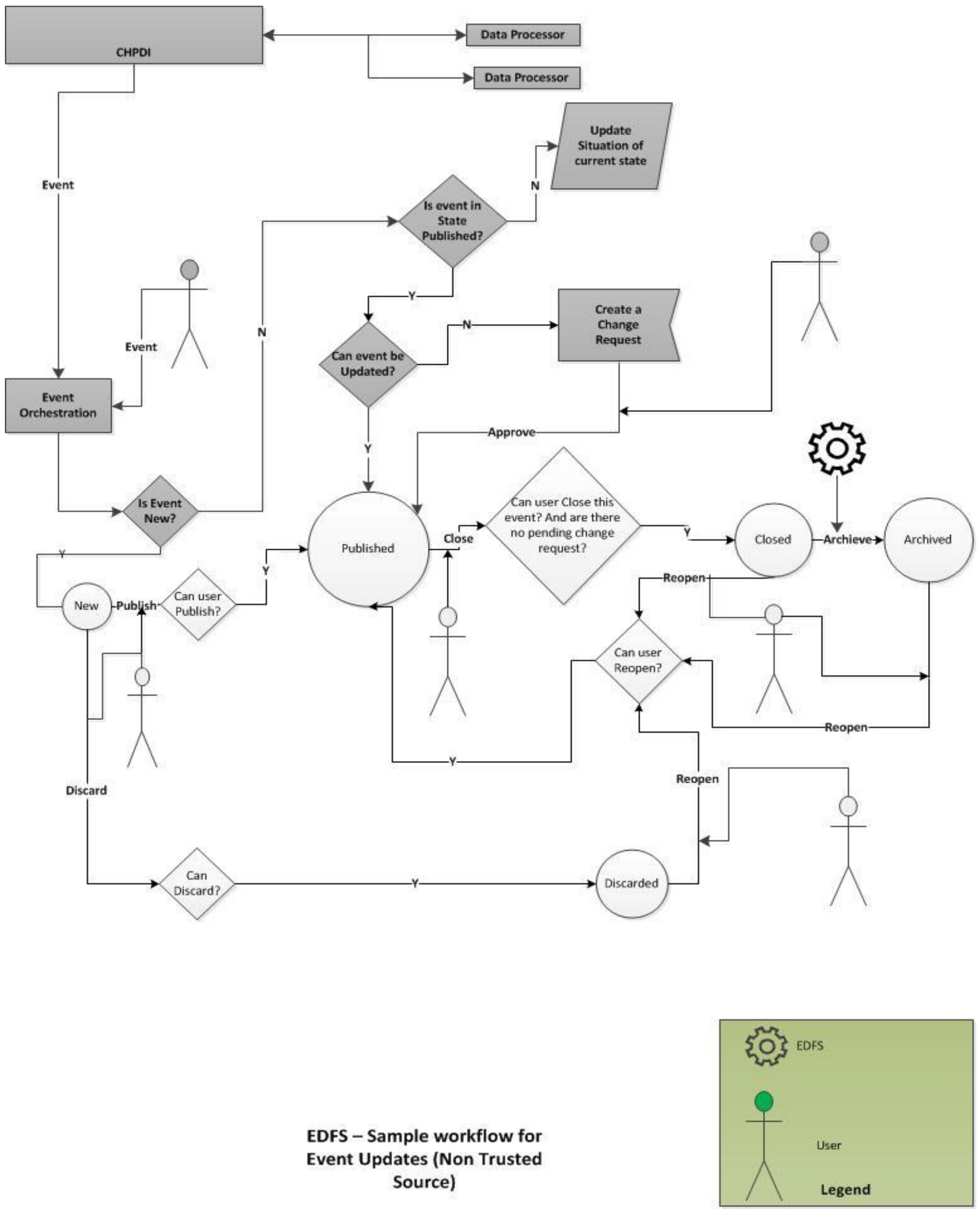


Diagram 3: Sample EDFS Workflow (Update Event)

6 System Architecture and Data flow

EDFS consists of multiple modules to minimize dependencies between subsystems, and also to introduce layers of abstraction. These modules are classified based on the type of functionality they perform and can be easily modified as needed without affecting other parts and subsystems. Since each module is independent of the other, updates or changes can be carried out without affecting the system as a whole. This allows for better maintainability and extensibility. The separation of these modules also allows for scalability e.g. having the web module not tightly integrated with the event orchestration module will allow the website to be easily scalable if needed.

At high level, EDFS comprises of ten core modules that are listed below:

6.1 Configuration Module (CM)

Configuration module provides necessary configuration parameters that are required by other modules in EDFS. The configuration module will provide methods to read and update the configuration store. CM is also responsible for notifying modules when configuration has been updated by system admins. A module will then invalidate its current set of configuration and reload the new set.

6.2 Audit Module (AM)

Audit module provides methods that can be used to submit audit trails to a database. This will also provide for methods that will be used by the reporting interface to build and display various audit reports.

6.3 Alarm Module (ALM)

Alarm module provides methods that can be used create and update alarms in a database. This will also provide for methods that will be used to create Audit records to track changes for alarm updates.

6.4 Logging Module (LM)

Logging module provides methods that can be used to submit log data to a database. Log levels will be configurable such as *debug* and *error* and may result in generating and sending notifications.

6.5 Publishing Module (PM)

This module provides methods for publishing data to external systems and will act as interface between EDFS and those external systems. Publishing module will consist of sub channels such as JMS channel, API etc.

6.6 Event Module (EM)

This module provides methods for managing event data within EDFS, such as creating and updating events, validation and interacting with the Audit and Alarm modules to track changes to an Event.

6.7 Workflow Module (WM)

This module provides functionality to manage event workflow from state to state. Workflow will be configured by EDFS administrators with each data source having multiple workflows. EDFS will use default system workflow if a workflow cannot be located for a specific type of event.

6.8 User Module (USM)
This module provides methods for authentication/authorization and user management methods such as creating new users, roles, assigning and revoking privileges.

6.9 Utility Module (UM)
This module provides various ancillary methods such as date/time conversion, data sanitizing and formatting, well-formedness checks etc. that can be used across different modules.

6.10 Database Module (DM)
This module acts as a bridge between the database and other modules. It provides basic data access methods and is designed such that the entire module can be easily modified or replaced if the underlying database is switched to another provider. For example, Audit module will make use of methods from database module to save audit logs, with the database module executing the actual SQL statements that will write those logs to a database.

EDFS consists of subcomponents such as web and the event orchestration engine that provide a runtime environment for various EDFS methods. The above listed core modules are internally consumed by these subcomponents to execute various operations. Listed below are the different subcomponents within EDFS.

6.11 Web Component
This provides all of the user interfaces for EDFS, data and system management. This component consists of various HTML/JavaScript pages that allow for all the user interactions with EDFS.

6.12 Data Interface (DI)
Events in EDFS can originate from multiple external sources. EDFS exposes a DI to each of these external data source. A DI is responsible for receiving, parsing, processing, and sanitizing incoming data.

DI internally makes use of Data Processors, explained below, to process and transform data received. For example, a data interface for CHP CAD may use *Geocoder Data Processor* to determine geographic location of an event. The sequence of DPs that a DI will pass messages through will be configurable using the EDFS system admin interface.

6.13 Data Processor
Data processors are modules that perform various methods such as geocoding and data sanitization. These modules will be primarily used by Data Interfaces. Additional data processors can be added to EDFS if a new requirement arises in future.

6.14 Event Orchestration Engine (EOE)
After data has been processed by a Data Interface using various Data Processors it will then be submitted to the orchestration engine which is where an event starts its lifecycle. EOE is responsible for managing an event’s lifecycle following a workflow or set of rules configured by administrators. Each event, based on its source type will have a defined workflow in the system.

6.15 Log Processor

This component handles the log persistence requirement for EDFS. Other components will make use of interfaces exposed by Log Processor to submit error and debug logs during various operational scenarios within EDFS.

6.16 Audit Processor

This component handles audit recordings within EDFS. Other components will make use of interfaces exposed by Audit Processor to submit audit logs during various operational scenarios within EDFS.

6.17 Alarm Processor

This component is responsible for monitoring state of events and generates alarms based on certain criteria specified in the requirements. The alarms are then displayed to users via the website.

6.18 Archive Processor

This component is responsible for archiving the closed events which are older than number of days (which can be configurable).

6.19 Publishing Processor

This component provides runtime environment and is responsible for publishing data to various external channels such as JMS, API, and Twitter etc. Publisher will be utilized by Event Orchestration Engine to publish an event when it's in "Published" state.

Note: Since once of the channels is JMS, a Java based client will be used to push to JMS topics and respond to JMS Queues. Software written using non-Java based JMS clients have limitations in error handling and do not implement all the specifications per JMS specifications.

The following diagram shows different modules and components defined above that together constitute new EDFS.

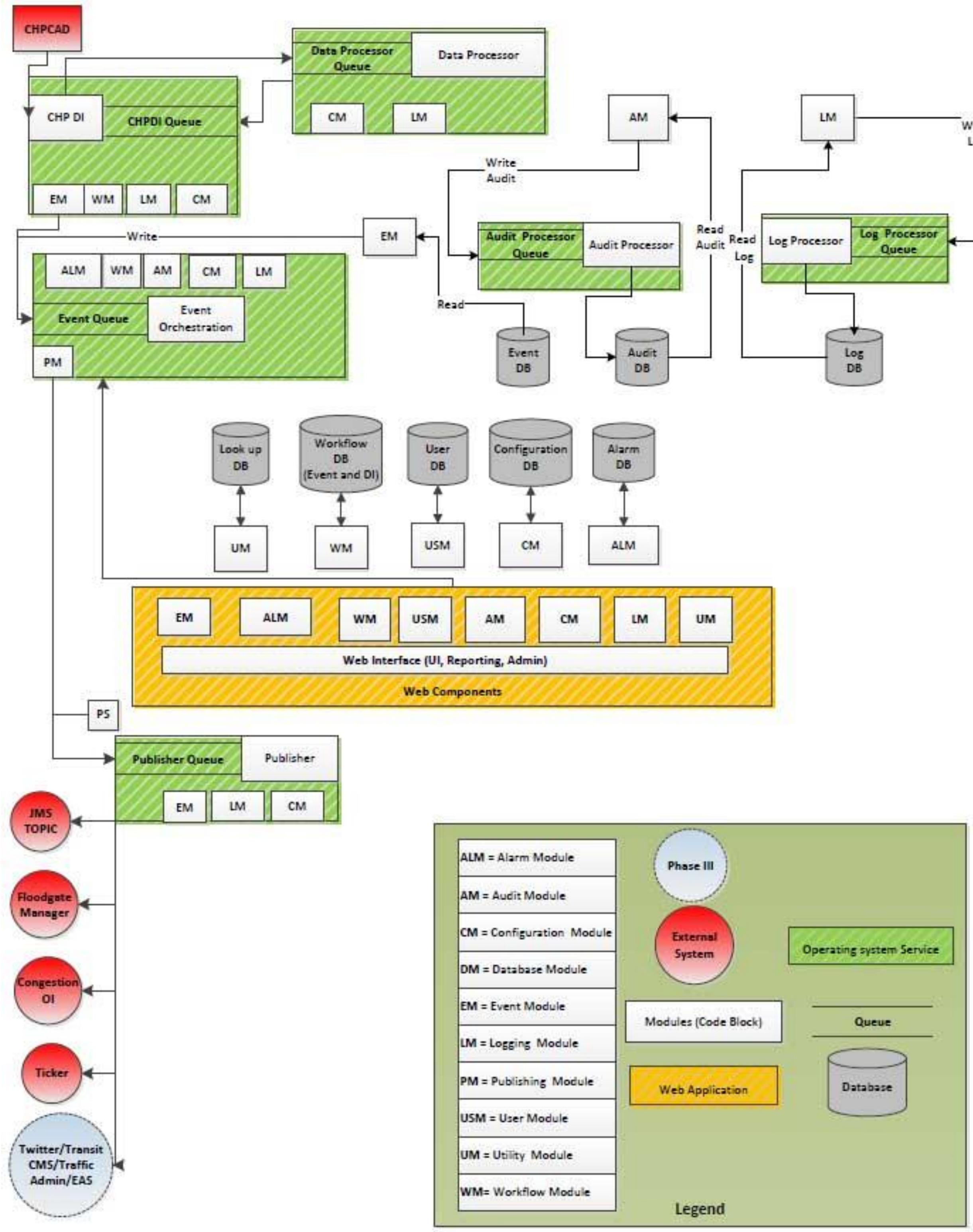
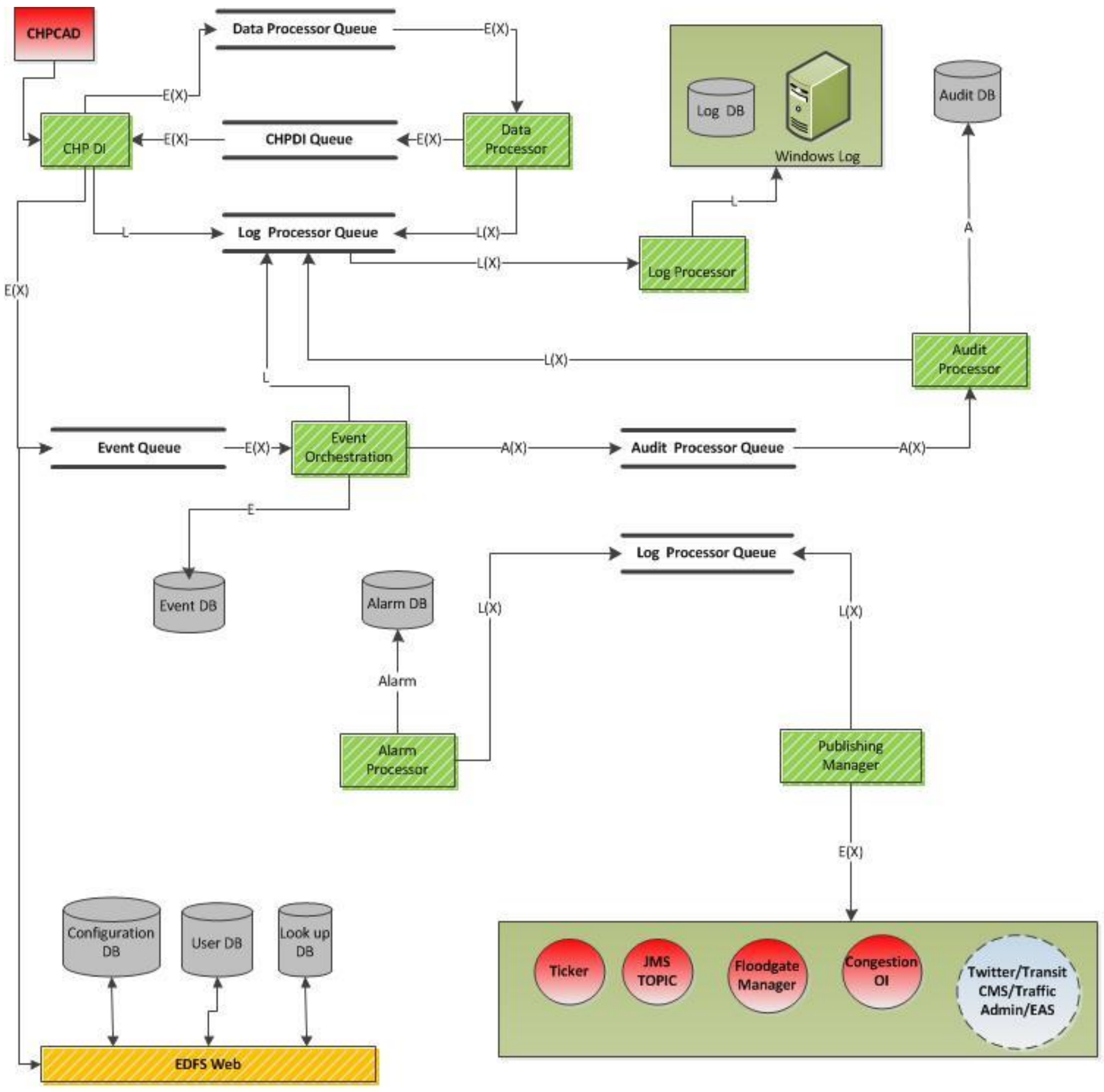


Diagram 4: EDFS Architecture

The following diagram shows the data flow between components of the system.



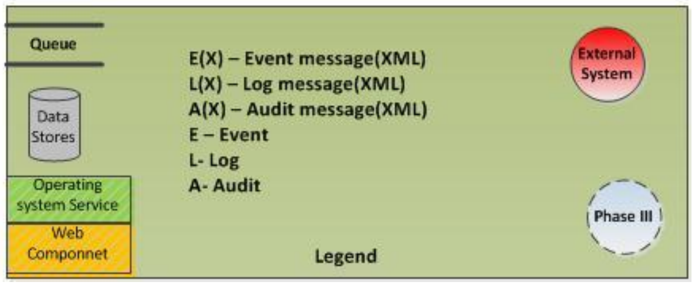


Diagram 5: EDFS Data Flow

6.20 Data Elements

EDFS is a data management tool which allows collection and manipulation of various data from different sources. Amongst these, Event is the core data with remaining data types providing additional features to help manage it. This section list and defines each of these data elements. Many of these elements will be represented by an individual data object in EDFS.

6.20.1 Event Information

Represents various traffic and transit related events, and incidents within the 511 system. Changes to an event will create an Event history entry and an Audit record.

Event Type	
EventTypeID	Type of event (e.g. “Traffic”, “Transit”, “Emergency”, etc.), see Event Type for possible values.
Event Sub Type	
EventSubTypeID	Sub Type for the selected Event Type. For “Traffic” would be “Incident” (which would include Accidents, Fire, Weather, Short-term Construction, Roadwork etc.), “Construction”, and “Special Event”. Other event types may have their own sub types defined. See Event Sub Type for possible values.
Traffic Sub Types	

CategoryID	For EventType “Traffic”:
	for SubType “Construction”: **see ConstructionTypes.csv**
	for SubType “Incident”: **see IncidentTypes.csv**
	for SubType “Special Event”: **see SpecialEventTypes.csv**

For Special Events	
SpecialEventDetails	
Event Location	
CountyID	
Latitude	**From MDB tblFacility->tblPointLocation by** CrossStreetStart **Facility_ID**
Longitude	**From MDB tblFacility->tblPointLocation by** CrossStreetStart **Facility_ID**
PostMiles	Postmile markers indicate the distance a route travels through a County. ie: “880 ALA 99” is mile 99 in Alameda County on Route 880.
SourceLocationDescription	e.g. "JWO"

RouteFacilityID **RouteFacilityName**	Route/Facility, Bridge and Venues **See MDB tblFacility** **Save Facility_ID in tblEvents, Facility_Name in history and archive**
Direction	
Descriptor	(e.g. "before", "after", "at", "between" etc.)
CrossStreetStartPointID **CrossStreetStartPointName**	**Point_ID From MDB tblFacility->tblPointLocation by Facility_ID** **Save Point_ID in tblEvents, Point_Name in history and archive**
CrossStreetEndPointID **CrossStreetEndPointName**	**Point_ID From MDB tblFacility->tblPointLocation by Facility_ID** **Save Point_ID in tblEvents, Point_Name in history and archive**
Impact	
PlayTypeID	(list of entries will be provided from existing EDFS)
TravelerAdviceID	(list of entries will be provided from existing EDFS)
PavementConditionID	Will only apply to incident type events. (list of entries will be provided from existing EDFS)
WeatherConditionID	Will only apply to incident type events. (list of entries will be provided from existing EDFS)
TravelerImpactID	(list of entries will be provided from existing EDFS)
ComputedScore	(algorithm will be provided by Bimesh)
Event Background Information	
ID	Unique identifier of an event within the EDFS system
DataSourceID	The Data Source of the event (e.g. "CHP CAD" etc.), see Data Source Type for possible values
ExternalID	Unique identifier provided by external data feed, used to associated incoming events with an existing event (a.k.a. “Event ID in Source’s System”)
UserID	Owner of the event
OriginalEventDescription	Original event description from source
ExpectedDelayTime	
ExpectedMilesOfBackup	
RelatedEventID	Unique identifier of the “parent” event that the event is related to
RelationshipTypeID	Type of relationship between an event and its RelatedEvent (e.g. "Child", Duplicate" etc.), see Event Relationship Type for possible values
PointOfContact	

PointOfContactPhone	
PointOfContactEmail	optional
Notes	
Description	Will be constructed from UI selections.
DateCreated	Date and time when event was created
LastUpdated	Date and time when event was updated
StartDate	*Min(Schedule.StartDate)*
EndDate	*Max(Schedule.End)*
Event Schedule	
For Long-Term construction and events that span over multiple days EDFS shall allow creating multiple schedules per event.	
Operators shall also be able to define Lane Impacts for each schedule entry.	
Number, Type, status and lane details of lanes affected(The list of entries will be provided from existing EDFS)	
Operators shall also be able to graphically define lane impacts. The underlying link definition should be used to construct lane diagram. In case where an incident impacts expands multiple exists and multiple links, the lane configuration for first link should be used. The operators can choose to define impact using list as well as graphical lane representation.	
StartDate	Scheduled Start Date and Time
EndDate	Scheduled End Date and Time
EstimatedDuration	Estimated event duration
LeadTime	How long before the StartDate should we publish the Event?
LagTime	How long after the EndDate should we close the Event?
Continuous	(not sure what this means yet)
ActiveDays	(All, Mon, Tue, Wed, Thru, Fri, Sat, Sun, Weekday, Weekend)
ImpactLevelID	(Major, Minor, Moderate, Severe)
PeakHourDelay	(list will be provided from EDFS)
AlternateRouteFacilityID **AlternateRouteFacilityName**	Route/Facility, Bridge and Venues **See MDB tblFacility** **Save Facility_ID in tblEvents, Facility_Name in history and archive**
PeakAlternateRouteDelay	
AffectedLaneCount	Number of lanes affected by the event
LaneTypeID	
LaneStatusID	
LaneDetailID	
System Information	

StateTypeID	Current state of the event (e.g. “New”, “Published”), see Event State Type for possible values
PublishingChannels	A comma delimited list of Dissemination Channel IDs. For automatically published events, this list will be all of the channels available for the data source. For user edited events, this will contain the selected channels for publication. There will be an Event Channel Publishing Status record for each entry in this list.
ReadyToPublishTimeStamp	Timestamp used to detect UI changes to an event during publishing. If the timestamp was changed during publishing, the publishing manager will NOT update the channel publishing statues after the event has published. The events need to be republished again.
RequestedActionTypeID	Action to be taken against the event by Event Orchestration (e.g. “Publish”, “Close”), see Action Type for possible values

6.20.1.1 *Event Types and Sub Types*
Defines basic type of an event, currently defined values are:

Event Type		**Sub Types**	
1	Traffic		
		1	Construction
		2	Emergency
		3	General
		2	Incident
		3	Special Event
2	Transit		

6.20.1.2 *Categories*
Defines classification of a “Traffic” event, currently defined values are:

- SubType “Construction” includes “Bridge Painting”, “Paving”, etc.
- SubType “Incident” includes “Earthquake”, “Fire”, etc.
- SubType “Special Event” includes “Baseball Game”, “Fleet Week”, etc.

6.20.1.3 *Impact Level Type*

Defines impact of an event on traffic lanes, currently defined values are:

Impact Level Type	
1	Major
2	Minor
3	Moderate
4	Severe

6.20.1.4 *Data Source Type*

Defines the data source of an event, currently defined values are:

Data Source Type		**Incoming data feed is automatically published (Could be changed as needed)**
1	EDFS User Interface	No
2	Caltrans LCS	Yes
3	Caltrans CWWP	Yes
4	CHP CAD	Yes
5	Events Venue Provider RSS/Atom feeds	No
6	Twitter	No
7	Structured Email Message	No
8	Transit Agency	Yes
9	Caltrans D-4 ATMS	Yes
10	Media	No
11	Email	No

6.20.1.5 *Event Relationship Type*

Defines type of relationship between an event and its RelatedEvent, currently defined values are:

Event Relationship Type	
1	Child
2	Duplicate
3	Other

6.20.1 Event Publishing

Events are selected for publishing based on the Dissemination Channels defined for the event, and the channel publishing status of each channel. Every schedule events have two records (publish or close) per schedule per channel. For instance, event ID 123 has three schedules and need to publish for 2 channels. Total entries in this table will be 2 records (publish and close) * 3 schedules * 2 channels = 12 records.

6.20.1.1 *Dissemination Channel Type*

Dissemination channels are used by the publisher service to send event information to external systems.

Dissemination Channel Type		Requires
1	511 Traffic Website – Breaking News (using Traffic Admin)	
2	511 Traffic Website – Construction (using Traffic Admin)	
3	511 Traffic Website – Traffic Map (Congestion and B-A-L using Congestion OI)	Supervisor privilege
4	Floodgates on 511 Phone (using Floodgate manager)	
5	Floodgate on MY 511 Phone (using Floodgate manager)	
6	511 Traffic Data Feed - TOMS	
7	Twitter Account - #twitter1 (example)	
8	Ticker	
9	511 Transit Website (using Transit CMS)	
10	511 Mobile Website	

6.20.1.2 *Event Channel Publishing Status*

Defines the publishing status for each dissemination channel of an event.

Event Publishing Status	
EventID	Links to an event
ScheduleID	Links to a specific event's schedule
EventPublishStatusType	See Event Publish Status Type for possible values
DisseminationChannel	See Dissemination Channel Type for possible values
ChannelPublishStatusType	See Channel Publish Status Type for possible values
ScheduleDate	Date and Time when an event needs to publish or close. The schedule date value will be the same as schedule's start date or end date based on EventPublishStatusType value.

6.20.1.3 *Event Publish Status Type*

Defines the event's publishing statues, currently defined values are:

Event Publish Status Type	
1	Publish
2	Close

6.20.1.4 *Channel Publish Status Type*
Defines the channels' publishing statuses, currently defined values are:

Channel Publish Status Type	
1	To be published
2	Was published
3	Error

6.20.2 User Information
Represents a user within the EDFS system, typically operators and supervisors. A user will have privileges on various resources within EDFS. A user will also have list of preferences.

User	
ID	Unique identifier of a user within the EDFS system
UserName	
UserEmail	
FirstName	
LastName	
StatusTypeID	The Status of this user (e.g. "Active" etc.), see User Status Type for possible values
ProfileStatusTypeID	The Profile Status of this user (e.g. " NeedToChangePassword " etc.), see User Profile Status Type for possible values. For instance, "NeedToChangePassword" flag force the user to change the password when he/she login.
RoleID	The Role that this user is assigned (e.g. "Operator" etc.), see Role Type for possible values. A user belongs to a single Role and inherits the privileges assigned to that Role.
FailedPasswordAttemptCount	Record the number of failure login attempts. Check this value against "MaxInvalidPasswordAttempts" configuration value and lock the user if the value is equal or greater.
FailedPasswordAttemptWindowStart	Record the last time login failed. Check this value against "MaxPasswordAttemptWindow" configuration value at next

	login and increase “FailedPasswordAttemptCount” value if the login attempts are within the window.
LastLogin	
DateCreated	
LastUpdated	

6.20.2.1 *User Status Type*
Defines the status of a user, currently defined values are:

User Status Type	
1	Active
2	Disabled
3	Locked

6.20.2.2 *User Profile Status Type*

Defines the profile status of a user, currently defined values are:

User Status Type	
1	None
2	NeedToChangePassword
3	NeedToRegister

6.20.2.3 *User Challenge Questions*

Represents the challenge questions and their answers that would be used when retrieving a user's password.

User Challenge Questions	
UserID	
ChallengeQuestion	e.g. "What is your Fathers middle name"
ChallengeAnswer	

6.20.2.4 *User Session*

Represents a user session within EDFS and maintains details such as authentication ticket, date session was created and last activity time.

User Session	
UserID	
AuthTicket	
ClientIP	
CreatedDate	
LastActivityTime	

6.20.3 User and Role Privileges

Represents privileges assigned to a data source and/or event type, along with the available actions and dissemination channels.

User and Role Privilege	
UserID	Required when assigning a privilege to an individual user
RoleID	Required when assigning a privilege to a role (instead of an individual user), see Role Type for possible values
ResourceType	See Resource Type for possible values

DataSource	Required if ResourceType is “Data Source”, see Data Source Type for possible values
DisseminationChannel	Required if ResourceType is “Dissemination Channel”, see Dissemination Channel Type for possible values
EventType	Required if ResourceType is “Event”, see Event Type for possible values
Privileges	Privileges assigned to either a Role or User will contain combinations of these values (e.g. Read+Publish+Close), see Privilege Type for possible combinations

6.20.3.1 *Resource Type*
Resources are objects within EDFS that have Privileges defined on them.

Resource Type	
1	Dissemination Channel
2	Data Source
3	Event
4	User

6.20.3.2 *Role Type*
Defines the role assigned to a user or process, currently defined values are:

Role Type	
1	EDFS (assigned to automated processes)
2	Operator
3	Supervisor
4	System Administrator

6.20.3.3 *Privilege Type*
Represents the privileges available on resources within EDFS. Privileges assigned to either a Role or User will contain combinations of these values (e.g. Read+Publish+Close).

Privilege Type	
0	None
Access privileges	
1	Read

2	Create
4	Update
8	Delete
Action Privileges	
16	Publish
32	Close
64	Discard
128	Archive

Example of privileges assigned to the Supervisor Role	
Dissemination Channel	**Privileges**
511 Traffic Website – Breaking News	Read+Create+Update
511 Traffic Website – Construction	Read+Create+Update
Data Source	
CHP CAD	Publish+Close+Discard
Event Type	
Incident	Publish+Close+Discard
User	Read+Create+Update+Delete

Example of privileges assigned to an individual user	
Dissemination Channel	**Privileges** (would override their Role privileges)
511 Traffic Website – Breaking News	Read
Data Source	
CHP CAD	Publish

6.20.4 Workflow

Represents a workflow defined for data originating from a particular source. Each workflow will consist of multiple states and transitions.

Example of CHP CAD Workflow			
State Stage	**From State**	**Transition Action**	**To State**
Begin	New	Publish	Published

		Close	Closed
		Discard	Discarded
Intermediate	Published	Update	Published
		Close	Closed
End	Closed	Discard	Discarded

6.20.4.1 *Workflow*

Controls the workflow for a data source.

Workflow	
ID	Unique identifier for a data source workflow
DataSource	See Data Source Type for possible values
Active	
DateCreated	

6.20.4.2 *Workflow State*

Defines the available “From States” for a workflow.

Workflow State	
WorkflowID	Workflow that this state belongs to
State	See Event State Type for possible values
StateStageType	The stage at which this state occurs in a workflow (e.g. “Begin”), see State Stage Type for possible values

6.20.4.3 *State Stage Type*
Defines the stages of a workflow process, currently defined values are:

State Stage Type	
1	Begin
2	Intermediate
3	End

6.20.4.4 *Transition*
Defines the actions that can be taken against an event in a specific state.

Transition	
WorkflowID	Workflow that this transition belongs to
Action	The action that will transition an event from one state to another state, see Action Type for possible values
FromState	The state that an event must be in to take the action, see Event State Type for possible values

6.20.4.5 *Action*
Defines the state that the event will transition into when taking a specific action.

Action	
ID	Value corresponds to an Action Type
Name	
TransitionToState	The state that an action will transition an event to, see Event State Type for possible values

6.20.4.6 *Action Type*
Defines the available actions that can be taken against an event.

Action Type	
1	Publish
2	Close
3	Update
4	Discard
5	Archive

6.20.4.7 *Event State Type*

Defines the possible states of an event.

Event State Type	
1	New
2	Published
3	Closed
4	Archived
5	Discarded

6.20.5 Alarms

Represents an alarm within the EDFS system. Changes to an alarm will create an Alarm history entry and an Audit record.

Alarm	
EventID	Links an alarm to an event
ExternalID	Links an alarm to an event by "Event ID in Source's System" (optional)
ScheduleID	Links an alarm to a specific event's schedule
UserID	The owner of this alarm (optional). If not set, the alarm will be global.
AlarmType	See Alarm Type for possible values
StatusType	See Alarm Status Type for possible values
Message	
StartDate	
IsGlobal	Indicates whether alarm is visible to all users
LastUpdated	

6.20.5.1 *Alarm Type*

Defines the types of alarms which can be created.

Alarm Type		
1	Error	
2	Start	Scheduled Event Start
3	End	Scheduled Event
4	Orphan	Orphan Event
5	Open	Open Event
6	General	

6.20.5.2 *Alarm Status Type*

Defines the possible alarm status values.

Alarm Type	
0	Inactive
1	Active

2	Snoozed
3	Confirmed
4	Pending

6.20.5.3 *Alarm Default Values*

Allows for configuration of default settings based on the event type and severity/impact.

Alarm Default Values	
EventType	The event type that this default value applies to, see Event Type for possible values
EventSubType	The event sub-type that this default value applies to, see Event Type for possible values
LeadTime	Number of minutes before event start date/time when this alarm will become active (e.g. 240 = 4 hours) For open event, this alarm will become active after event was published
LagTime	Number of minutes before event end date/time when this alarm will become active (e.g. 240 = 4 hours) This value is always NULL for unscheduled events
StartMessage	Message/Note for scheduled “start” and open event alarms
EndMessage	Message/Note for only scheduled “end” alarms
Enable	On/Off
Sound	Indicate whether the sound alarm or not
SnoozeInterval	Number of minutes this alarm is snoozing
LastUpdated	

6.20.6 Auditing

Represents an audit record within the EDFS system, and contains current and previous values for a changed field.

6.20.6.1 *Audit Record*

An audit record is created for each field changed when an alarm or event is updated.

Audit Record	
EventID	Links an audit record to an event
ExternalID	Links an audit record to an event by "Event ID in Source's System" (optional)
RecordType	Identifies whether an audit record is for an alarm or event, see Audit Record Type for possible values
HistoryID	Links this audit record to either an Alarm history or Event history record, depending on RecordType
FieldName	Name of the field that was changed
FromValue	Previous value
ToValue	Current value
TimeStamp	Date and time when the field was changed (for reporting)
UserID	ID of the user who made the changes

6.20.6.2 *Audit Record Type*

Identifies whether an audit record is for an alarm or event.

Audit Record Type	
1	Alarm
2	Event

6.20.7 Logging

A log message is created when an error condition occurs and should provide a level of detail required for debugging the cause of the error.

6.20.7.1 *Log Message*

Represents a log message within the EDFS system. The LogLevel determine the level of detail to include, and any follow-up actions required (e.g. sending a notification email). The LogLevel assigned to a specific Component will be configurable using EDFS system admin interface.

Log Message	
UserID	Links a log message to a user (optional)
EventID	Links a log message to an event (optional)
ExternalID	Links a log message to an event by "Event ID in Source's System" (optional)
ModuleType	Identifies the system module that created this log message, see Module Type for possible values
LogLevel	Identifies the logging level of this message and will contain combinations of these values (e.g. Message+Notification),see Log Level Type for possible combinations
Source	Identifies the location within the source code where this log message was generated (for debugging purposes)
Message	
StackTrace	Detailed information what would help in debugging the cause of an error. Requires setting the LogLevel to "Stack Trace" in the system configuration.
TimeStamp	Date and time when this message was logged (for reporting)

6.20.7.2 *Log Level Type*

The LogLevel assigned to a log message will contain combinations of these values (e.g. Message+Notification).

Log Level Type	
0	None
1	Message
2	Stack Trace
3	Notification

6.20.8 Configuration

This represents a list of configurable modules and a collection of configuration parameters per module within the EDFS system.

Below is an example of the configuration for the LogProcessor component having a Logging module. The component uses its MessageQueueName to listen for LogMessages and configuration change messages, and the module has a ConnectionString for persisting those messages to the database.

Example configuration for Log Processor Component			
ConfigurableObjectID	**ConfigurableObjectName**	**ParamName**	**ParamValue**
1	LogProcessor	MessageQueueName	“EDFS_Logging”
2	LogModule	MessageQueueName	“EDFS_Logging”

6.20.8.1 *Configurable Object Type*

Configurable Object Type	
ID	**Name**
1	Component
2	Module

6.20.8.2 *Configurable Component Type*

Configurable Component Type	
ID	**Name**
1	Audit Processor
2	Data Interface
3	Data Processor
4	Event Orchestration
5	Log Processor
6	Publisher
7	Website
8	Archive Processor
9	Alarm Processor

6.20.8.3 *Configurable Module Type*

Configurable Module Type	
ID	**Name**
1	Alarm
2	Audit
3	Database
4	Event
5	Logging
6	Publishing
7	Utility
8	Workflow

6.20.8.4 *Component Modules*

Identifies the Modules used by a Component.

Component Modules	
ComponentID	Identifies the Component, see Configurable Component Type for possible values
ModuleID	Identifies the Modules used by a Component, see Configurable Module Type for possible values

6.20.8.5 *Configuration Parameter*

Contains a single configuration parameter for either a Component or Module used by a Component.

Configuration Parameter	
ConfigurableObjectType	Either Component or Module
ConfigurableObjectID	If object type is Component will link to a Configurable Component Type, otherwise will link to a Configurable Module Type
Name	Name of the parameter (e.g. “LogLevel”)
Value	Value of the parameter (e.g. “3”)

6.20.8.6 *Configuration Change Message*

Sent to a components work queue when changing the configuration of the Component or a Module used by a Component

Configuration Change Message	
ConfigurableObjectType	Either Component or Module
ConfigurableObjectName	Identifies the Component or Module to which this configuration change applies, used to fine-tune the configuration change process to limit the impact of a change to only the affected object
TimeStamp	Date and time when the configuration was changed

6.21 Database Layer

This layer comprises of databases required by EDFS application and includes tables and procedures to store and manage data used by the various modules in EDFS. Listed below are the different databases and tables used. Please note, database diagrams have not been provided, as they would be prohibitively large.

6.21.1 EDFS Database

This database contains tables and procedures for events, users, and workflow.

6.21.1.1 *EDFS Database Tables*

Below are the tables within the EDFS database.

Table	**Data Object**
Event Information	
tblEvents	Event
tblEventSchedules	Event Schedules
tblEventEdits	*Tracks the user making an edit to an event and whether they selected the Publish action*
tblEventTypes	Event Type
tblEventCategories	Event Categories
~~tblEventSubCategories~~	~~Event SubCategories~~
tblDescriptors	
~~tblLaneImpactTypes~~	~~Lane Impact Type~~
~~tblLaneStatusTypes~~	~~Lane Status Type~~
~~tblLaneTypes~~	~~Lane Type~~
~~tblRampTypes~~	~~Ramp Type~~
~~tblShoulderTypes~~	~~Shoulder Type~~
tblDataSources	Data Source Type
tblEventRelationshipTypes	Event Relationship Type
tblDisseminationChannels	Dissemination Channel Type
tblDataSourceDisseminationChannels	*Available channels for each data source*
tblEventChannelPublishingStatus	Event Channel Publishing Status
tblPublishStatusType	Publish Status Type
History	
tblAlarmHistory	*Contains copy of tblAlarms item after each edit*
tblEventHistory	*Contains copy of tblEvents item after each edit*

tblEventScheduleHistory	*Contains copy of tblEventSchedules items after each edit*
tblServiceActivationHistory	*Contains audit trail of windows service activations*
Archive	
tblAlarmHistory_Archive	
tblAlarms_Archive	
tblAuditRecords_Archive	
tblEventHistory_Archive	Archived copy of tblEventHistory
tblEvents_Archive	Archived copy of tblEvents
tblEventSchedules_Archive	Archived copy of tblEventSchedules
tblEventScheduleHistory_Archive	Archived copy of tblEventScheduleHistory
tblLogMessages_Archive	
Unhandled Events	
tblUnhandledEvents	*Contains events received from message queue that could not be processed because of an error condition*
tblUnhandledEventSchedules	*Contains event schedule information received from message queue that could not be processed because of an error condition*
User Information	
tblUsers	User
tblUserStatusTypes	User Status Type
tblUserChallengeQA	User Challenge Questions
tblChallengeQuestions	*Contains list of challenge questions available to all users*
tblUserSessions	User Session
User and Role Privileges	
tblResourceTypes	Resource Type
tblRoles	Role Type
tblPrivilegeTypes	Privilege Type
tblUserPrivileges	*Privileges assigned to a Role*
tblRolePrivileges	*Privileges assigned to an individual User*
Workflow	
tblActions	Action Type

tblStates	Event State Type
tblStateStageTypes	State Stage Type
tblTransitions	Transition
tblWorkflows	Workflow
tblWorkflowStates	Workflow State
Alarms	
tblAlarms	Alarm
tblAlarmTypes	Alarm Type
tblAlarmStatus	Alarm Status Type
tblAlarmDefaults	Alarm Default Value
Auditing	
tblAuditRecords	Audit Record
tblAuditRecordTypes	Audit Record Types
Logging	
tblLogLevels	Log Level Type
tblLogMessages	Log Message
Configuration	
tblConfigurableObjectTypes	Configurable Object Type
tblConfigurableComponents	Configurable Component Type
tblConfigurableModules	Configurable Module Type
tblComponentModules	Component Modules
tblConfigParameters	Configuration Parameter

6.21 EDFS Modules

This section describes each of the EDFS modules in detail. The design and development of EDFS will follow a modular/component driven approach. These modules are structured following the “separation of concern” principle which also allows for a module to be fully designed and implement independent to other modules.

6.21.1 User Module

User Module provides for all things “User” within EDFS. Other modules will use User Module manage users, roles and their privileges.

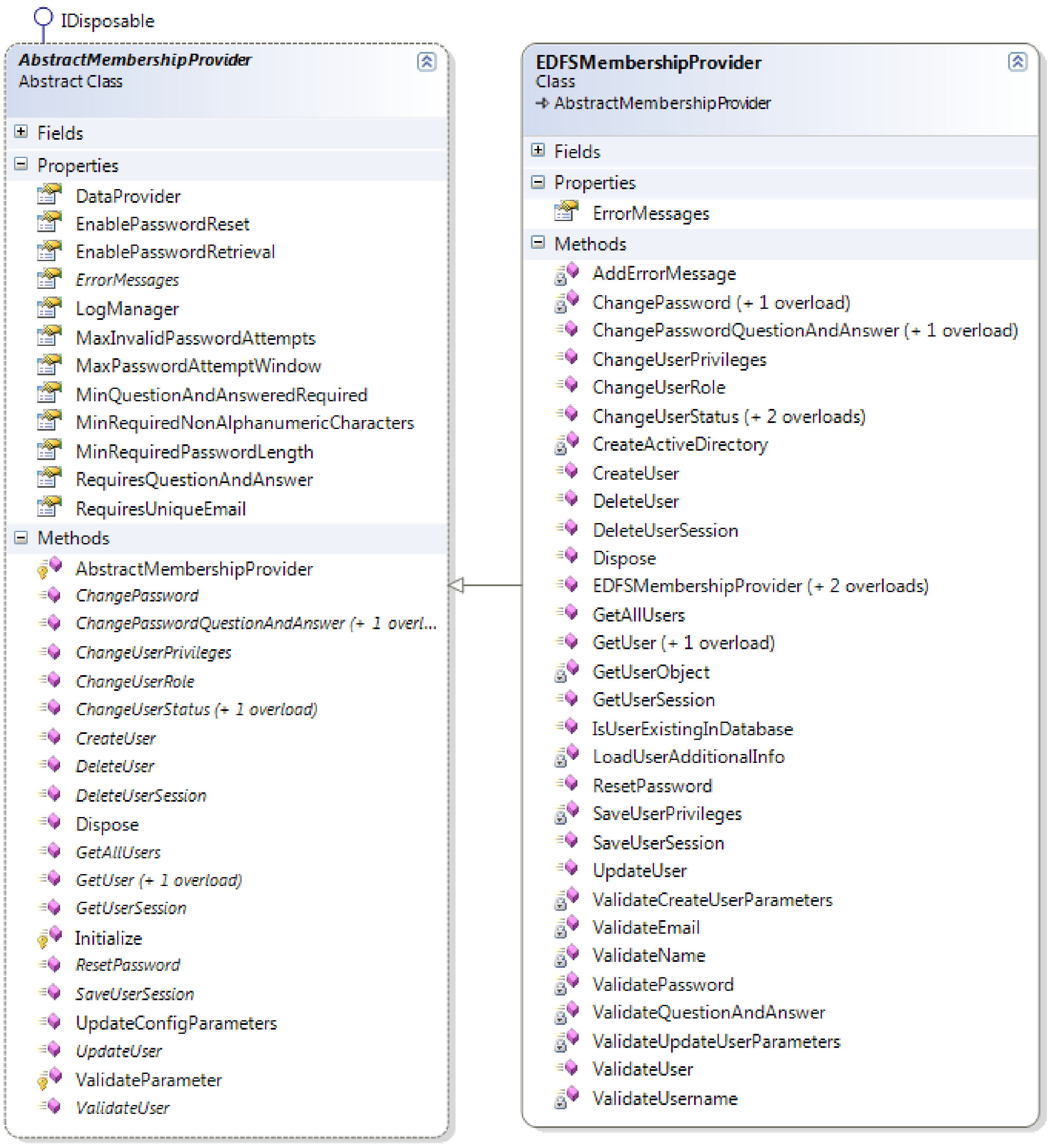
Methods performed by user module are:

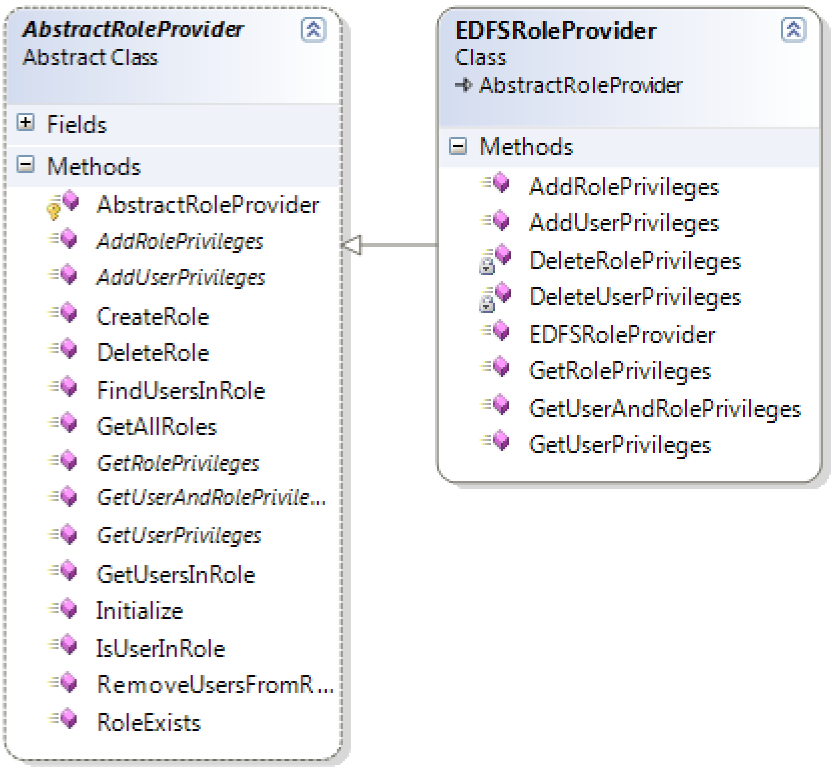
- Provide methods to create update and delete users, preferences, groups/roles and privileges.

- Provide methods to authenticate and authorize a user.
- Provide methods to manage user session within EDFS.

Two core classes, EDFSMembershipProvider and EDFSRoleProvider provide for all the above major functionalities. EDFSMembershipProvider inherits from AbstractMembershipShipProvider and EDFSRoleProvider inherits from AbstractRoleProvider. This layer of abstraction allows for easy extension and change in future, if such a need arises.

Below is a class diagram showing methods and properties available in User Module.





The primary consumer of user module is Web Component; different pages within EDFS website will make use of methods from EDFSMembershipProvider and EDFSRoleProvider.

For example, the login page will call Validate method to authenticate and authorize a user.

User module internally will call Authenticate method from ActiveDirectoryManager which upon successful validation will return a user identifier such as email back to user module. The User Module will then make use of Data Module to perform data user lookup and return a User object back to Web Component.

The various database tables used by Data Module to maintain data related to User Module are listed under EDFS database tables.

The diagram below shows dependencies between User Module and other modules within EDFS.

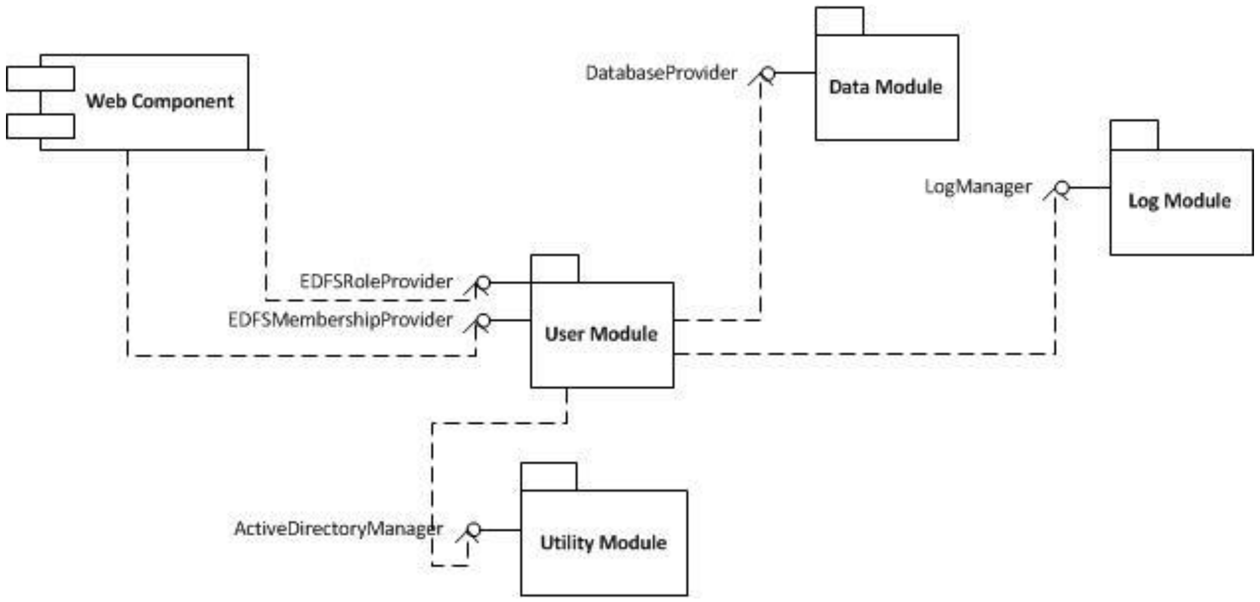
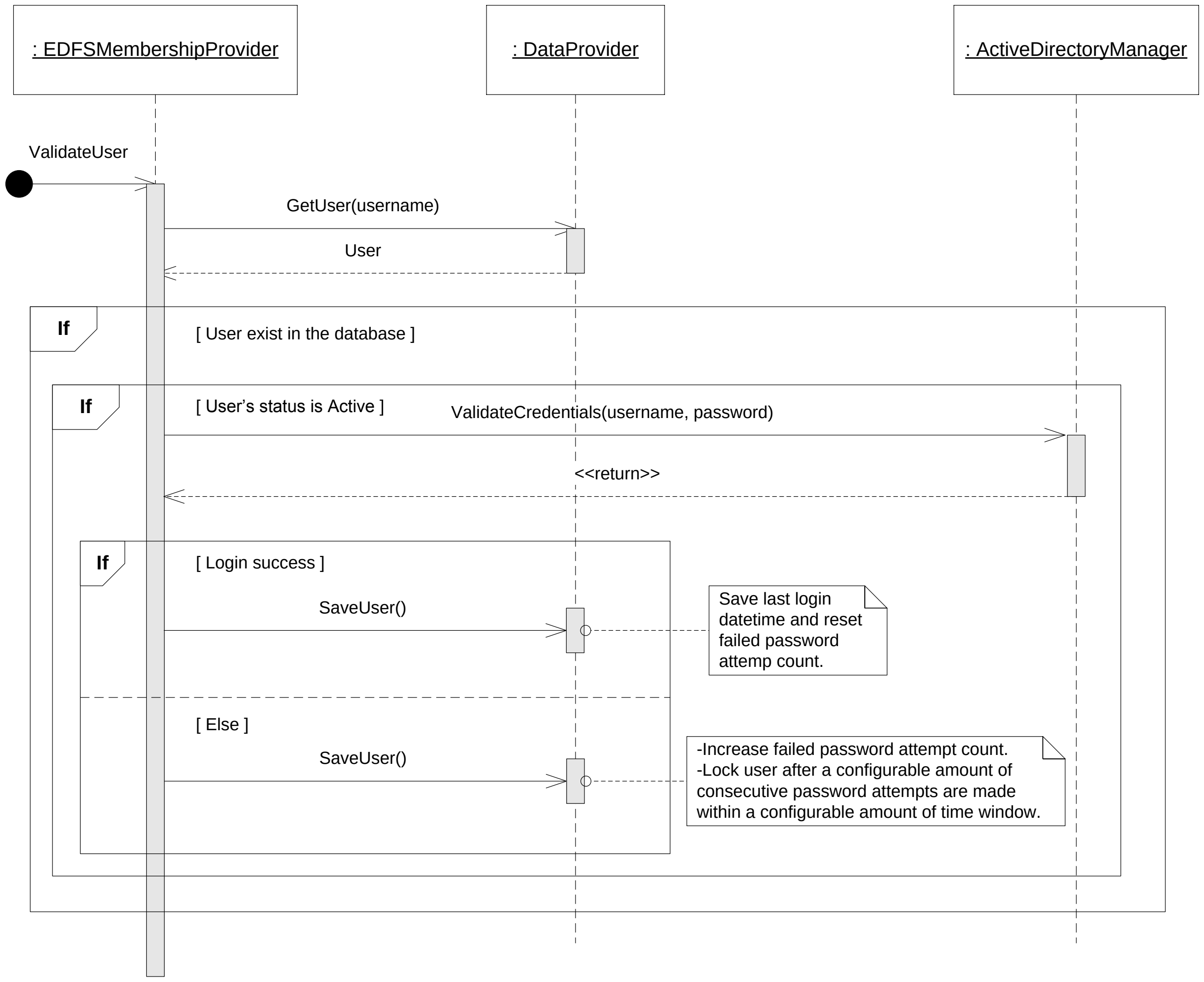


Diagram 5: User Module Dependencies

6.21.1.1 *Example of user account validation*

The validate user method checks if the user exists in the database. If the user exists, check the user's status to determine if it is Active or not. Only the Active can login to the system. This method will lock the user account after a configurable amount of consecutive password attempts are made within a configurable amount of time window. The last login date and time will be saved into database if the user login successfully.

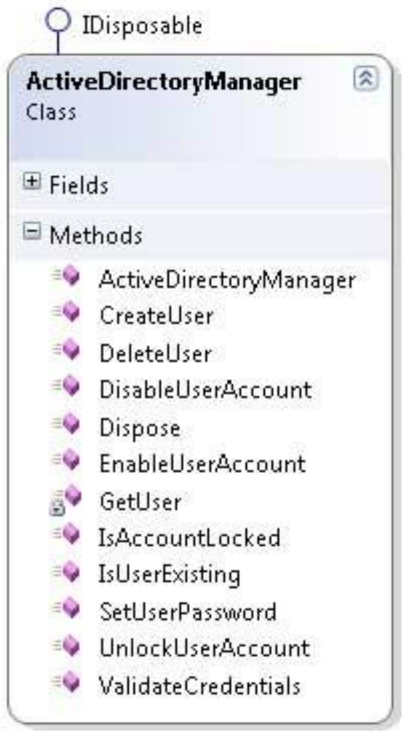


: EDFSMembershipProvider
: DataProvider
: ActiveDirectoryManager
ValidateUser
GetUser(username)
User
If
[User exist in the database]
If
[User's status is Active]
ValidateCredentials(username, password)
<<return>>
If
[Login success]
SaveUser()
Save last login datetime and reset failed password attemp count.
[Else]
SaveUser()
-Increase failed password attempt count.
-Lock user after a configurable amount of consecutive password attempts are made within a configurable amount of time window.

6.21.2 Utility Module

Utility module consists of classes and method that provide for wide variety of functions and are used across modules and components. For example, the ActiveDirectoryManager class provides methods that are used by User Module to perform ActiveDirectoy related methods such as validating a user, creating new user and roles within EDFS. The utility module does not provide a layer of abstraction as is the case with other modules. Most of the classes within utility module will be of type Static.

Below is the class diagram for ActiveDirectoryManager, which provides for various methods to interact with active directory.



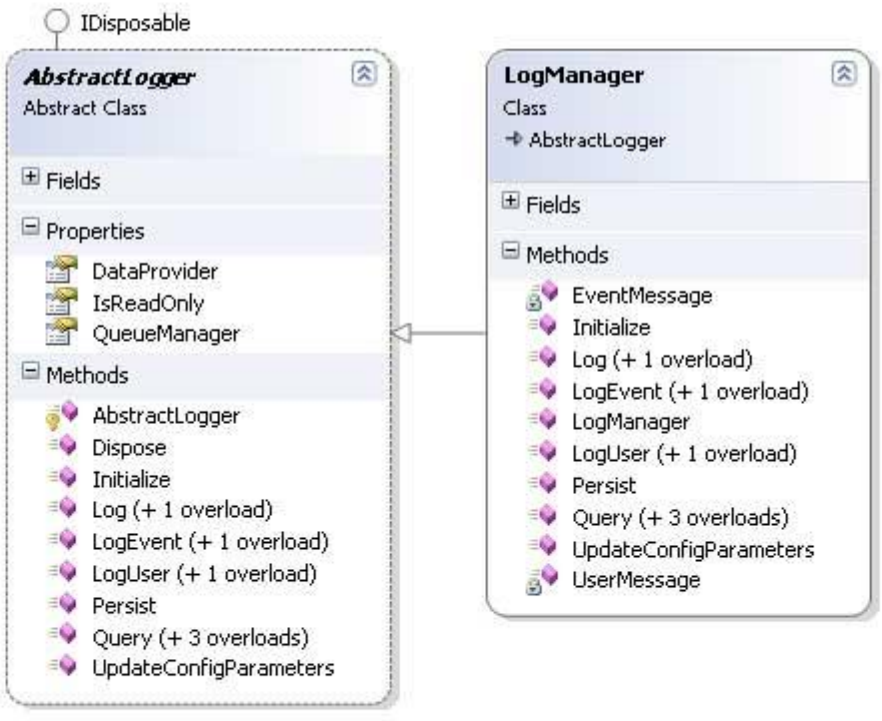
6.21.3 Log Module

Methods primarily performed by the log module are:

- Asynchronously log error messages to a queue (MSMQ) using the LogLevel of the LogCondition to determine the level of detail to log, and whether to send an email notification.
- Provide a method to write a queue message to persistent storage, called by the Log Processor service.
- Provide methods to query persistent storage based on event or date range criteria.

Below is the class diagram showing methods and properties available in the Log Module.

The AbstractLogger provides a common interface for all error logging within the EDFS system, and the LogManager provides an implementation of the AbstractLogger to read and write messages to a message queue and persistent storage.



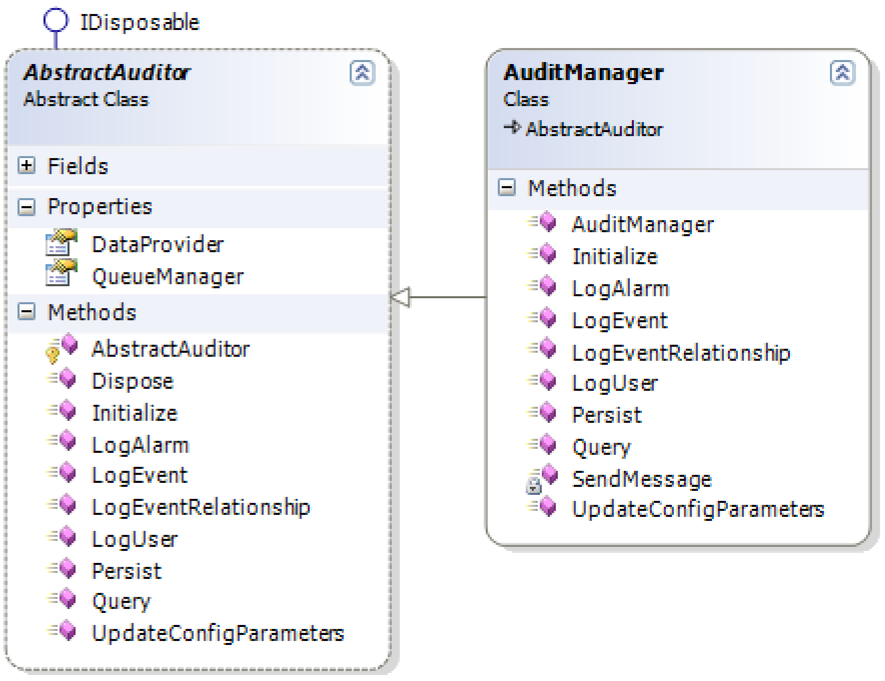
6.21.4 Audit Module

Methods primarily performed by the audit module are:

- Asynchronously log audit records to a queue (MSMQ).
- Provide a method to write a queue message to persistent storage, called by the Audit Processor.
- Provide methods to query persistent storage based on record type criteria.

Below is the class diagram showing methods and properties available in the Audit Module.

The AbstractAuditor provides a common interface for all auditing within the EDFS system and the AuditManager provides an implementation of the AbstractAuditor to read and write audit records to a message queue and persistent storage.

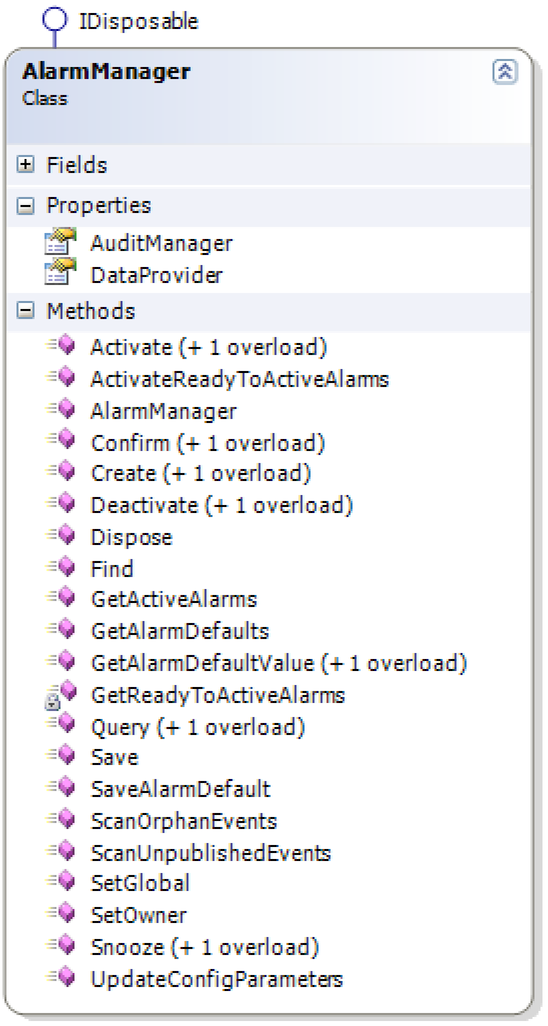


6.21.5 Alarm Module

Functions primarily performed by the alarm module are:

- Provide methods to create and update the state of an alarm using the DataModule to write to persistent storage, and AuditModule to track changes.
- Provide methods to query persistent storage based on Event ID criteria.
- Provide methods to query alarm default values based on Event Type and Event SubType.

Below is the class diagram showing methods and properties available in the Alarm Module.



6.21.5.1 *Processing a configuration change notification*

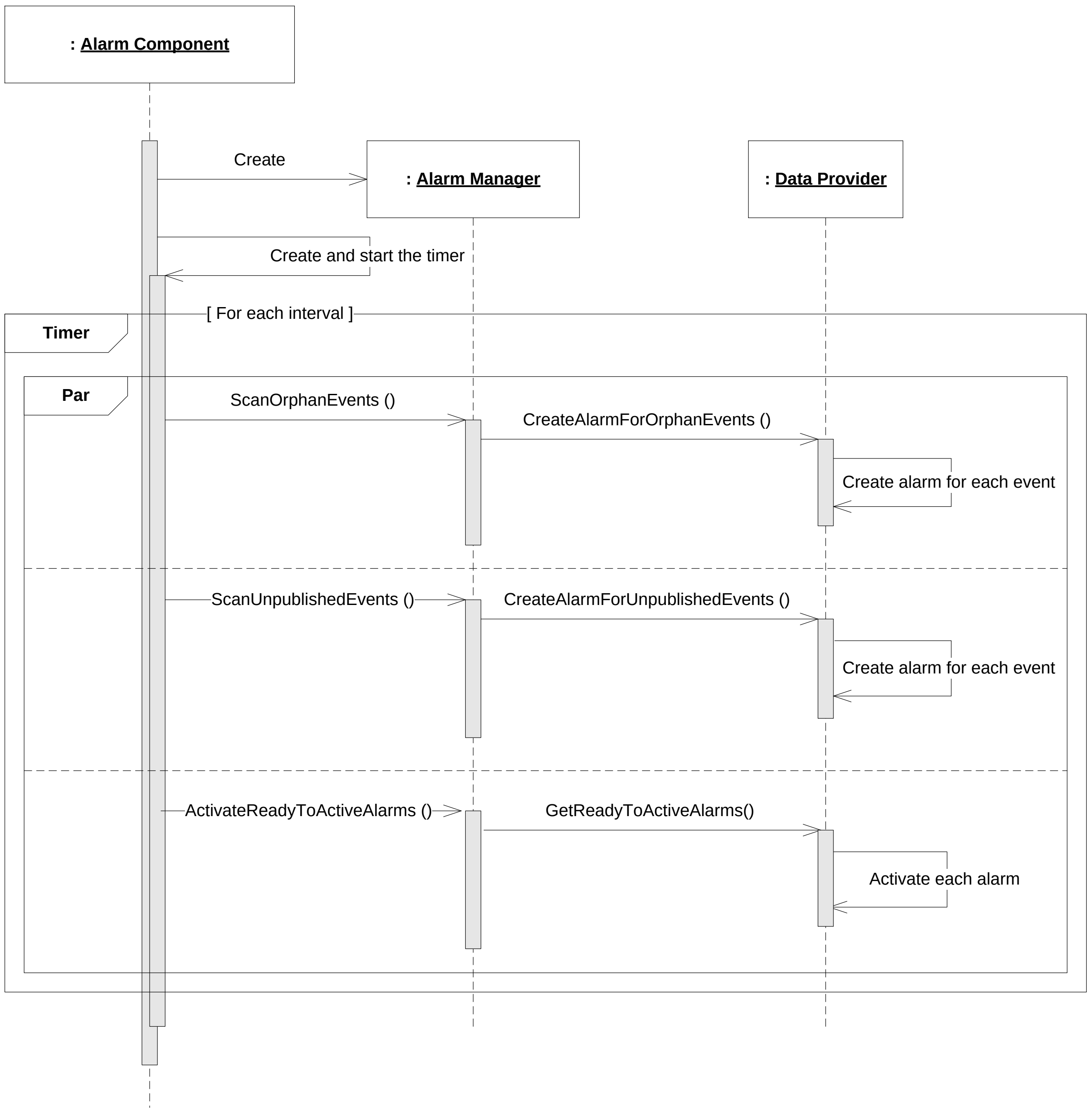
The alarm processor component inherits from AbstractEDFSComponent which provides the stream line for the configuration change notification.

When the listener receives a change message, it would use cross-thread communication to notify the main thread to pause after processing the currently running threads which are created by the component, stop the timer, retrieve the updated configuration parameters, update the configurations, reset the timer and resume processing.

The following modules and classes are involved in configuration updates:

1. Alarm module,
2. Audit module,
3. Data module and
4. Log module

6.21.5.2 *Example of alarm component process*



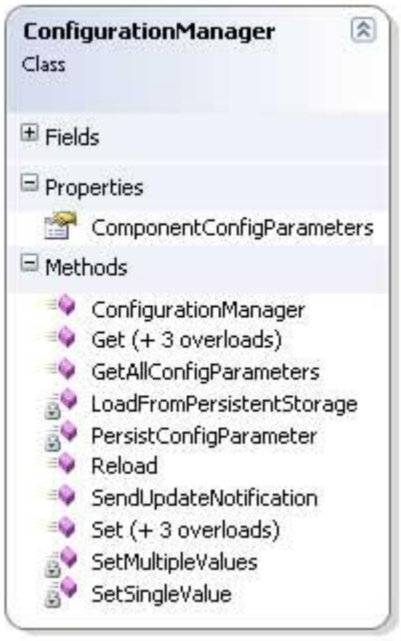
: Alarm Component
Create
: Alarm Manager
: Data Provider
Create and start the timer
[For each interval]
Timer
Par
ScanOrphanEvents ()
CreateAlarmForOrphanEvents ()
Create alarm for each event
ScanUnpublishedEvents ()
CreateAlarmForUnpublishedEvents ()
Create alarm for each event
ActivateReadyToActiveAlarms ()
GetReadyToActiveAlarms()
Activate each alarm

6.21.6 Configuration Module

Functions primarily performed by the configuration module are:

- Store configuration information in the database and provide methods to retrieve and update individual configuration parameters for each Component and its Modules.
- Apply configuration changes without restarting software or hardware and affecting users logged in by notifying affected Components and their Modules of a configuration change via the Components' message queue.

Below is the class diagram showing methods and properties available in the Configuration Module.



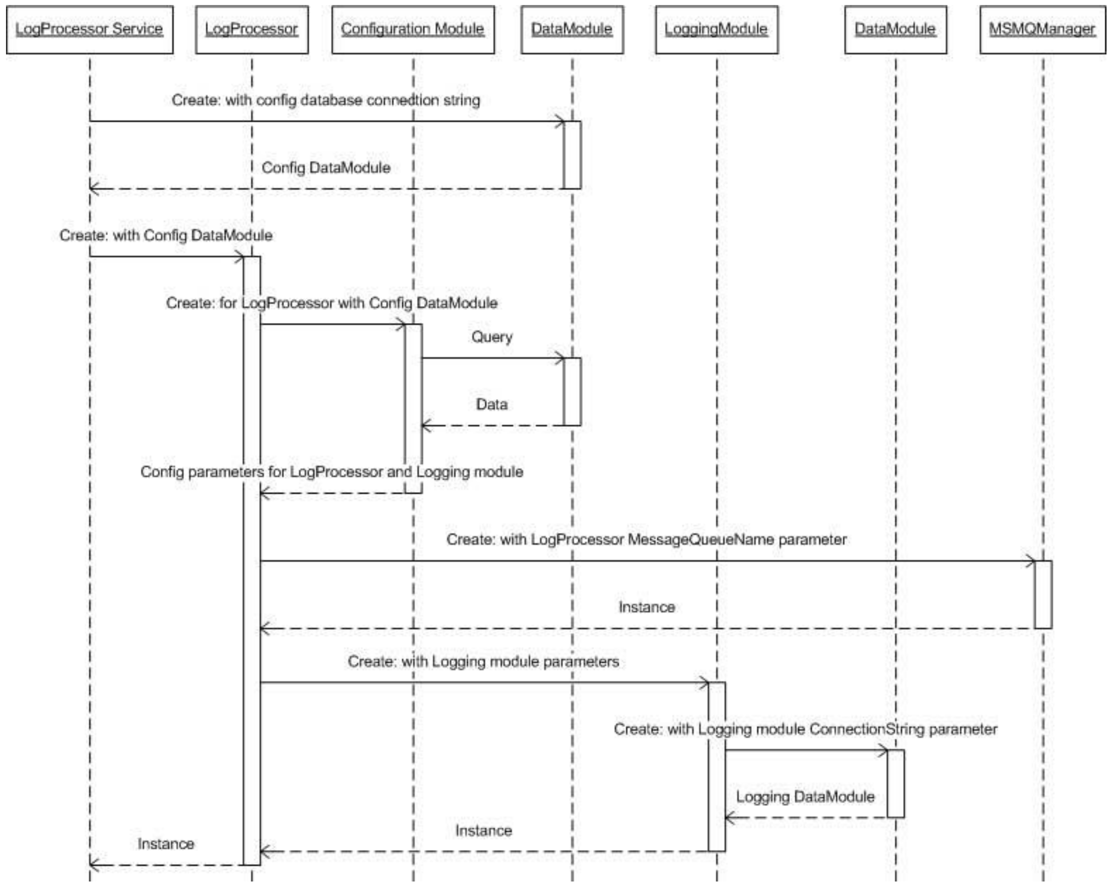
6.21.6.1 *Example of loading configuration parameters for the LogProcessor component*

The component constructor receives a DataModule instance pointing to the configuration database, and uses this to create an instance of the ConfigurationManager which calls GetComponentConfigurationByName('LogProcessor') returning the following data:

ConfigurableObjectID	ConfigurableObjectName	ParamName	ParamValue
1	LogProcessor	MessageQueueName	"EDFS_Logging"
2	LogModule	ConnectionString	"server=localhost;etc.."

The LogProcessor creates an instance of MSMQManager using its MessageQueueName to listen for Log Messages and configuration change messages.

The LogModule parameters are passed to the constructor for LogModule which creates a DataModule instance using the connection string for writing to the logging database. This creation pattern applies to all service components.

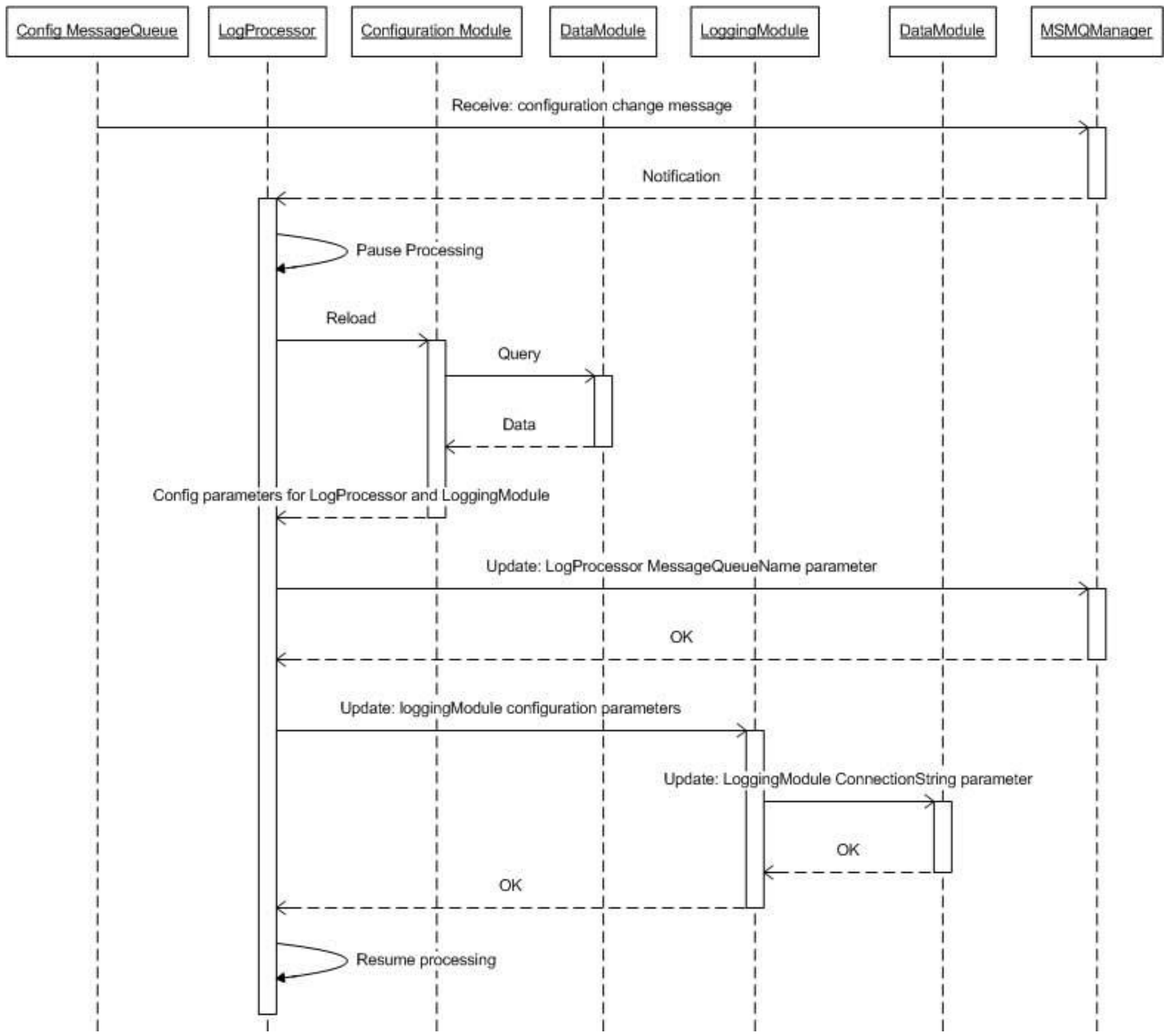

LogProcessor Service
LogProcessor
Configuration Module
DataModule
LoggingModule
DataModule
MSMQManager
Create: with config database connection string
Config DataModule
Create: with Config DataModule
Create: for LogProcessor with Config DataModule
Query
Data
Config parameters for LogProcessor and Logging module
Create: with LogProcessor MessageQueueName parameter
Instance
Create: with Logging module parameters
Create: with Logging module ConnectionString parameter
Logging DataModule
Instance
Instance

6.21.6.2 *Processing a configuration change notification*

For Components, the QueueListener would be a part of the component, running on a secondary thread. When the listener receives a change message, it would use cross-thread communication to notify the main thread to pause after processing the current record or batch, retrieve the updated configuration parameters and resume processing.

For web applications, a QueueListener service would receive the change message and update the timestamp on the file used for cache dependency, forcing the application to reload and cache the updated configuration parameters.

6.21.6.3 *Example of processing a configuration change notification for the LogProcessor component*


Config MessageQueue
LogProcessor
Configuration Module
DataModule
LoggingModule
DataModule
MSMQManager
Receive: configuration change message
Notification
Pause Processing
Reload
Query
Data
Config parameters for LogProcessor and LoggingModule
Update: LogProcessor MessageQueueName parameter
OK
Update: loggingModule configuration parameters
Update: LoggingModule ConnectionString parameter
OK
OK
Resume processing

6.21.7 Publishing Module

Funstions primarily performed by the publishing module are:

- Provide methods to create and register channels and publish (scheduled and unscheduled) events to them.
- Provide a method to move to “Publish” state from “New” state for scheduled events which need to publish right at the moment.

The AbstractChannel provides a common interface for all channels within the EDFS system. Push method provides common behavior logic for all channels. Drive classes must implement abstract methods such as Connect, Disconnect, Send, etc.

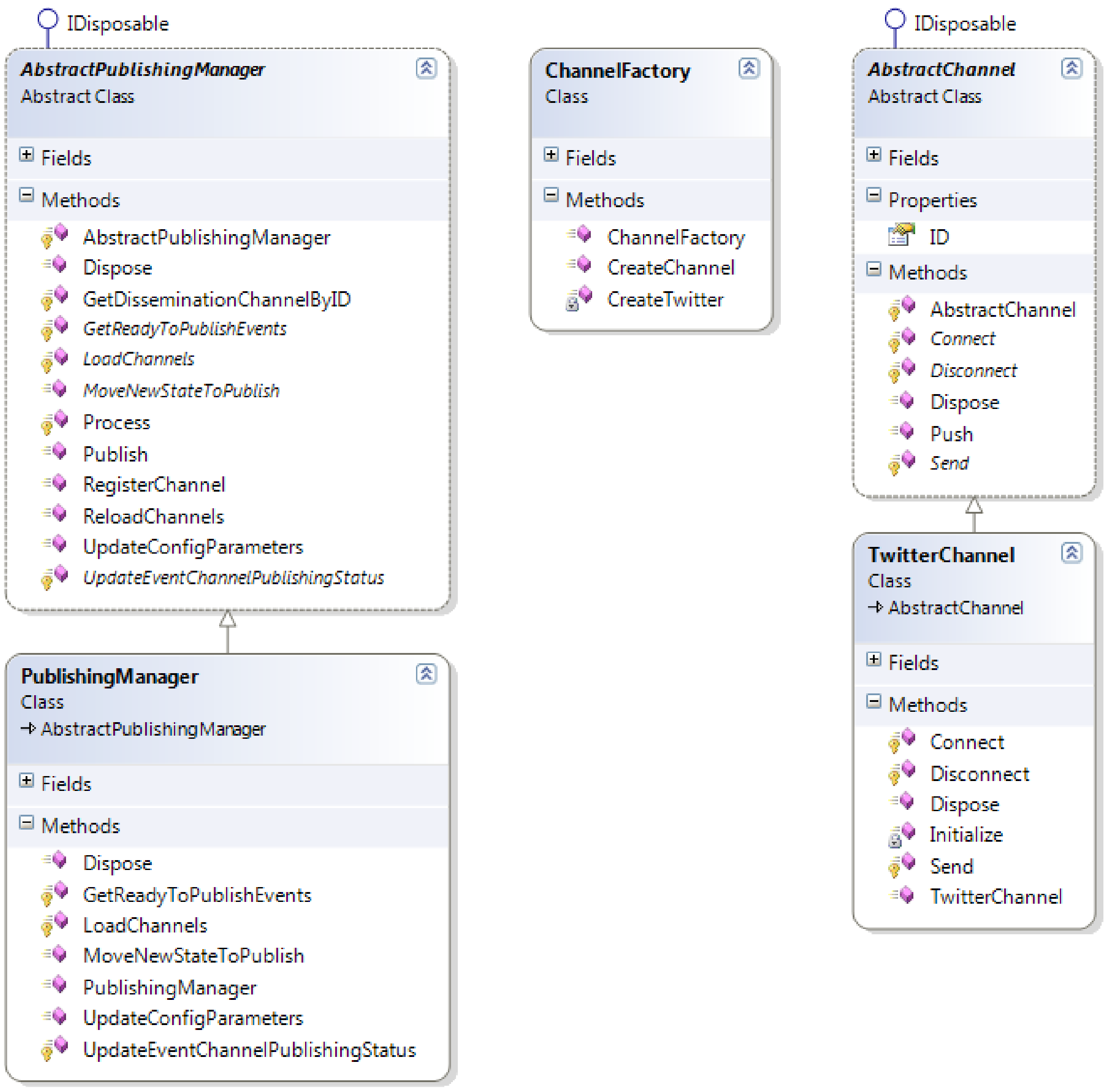
The AbstractPublishingManager provide a common interface for publishing the events and creating channels. Publish method provides common behavior logic for publishing events and updating the status to persistent storage. Drive class must implement abstract methods such as GetReadyToPublishEvents, CreateChannels, MoveNewStateToPublish, etc.

The PublishingManager provides an implementation of the AbstractPublishingManager to read the events from persistent storage and use ChannelFactory class to create channels.

The publishing component provides the following functionalities for publishing manager module:

1. Listening to (“EDFS_Publishing”) queue for update configuration changes which is implemented in AbstractEDFSComponent class.
2. Provide timer to execute the publishing manager’s methods.

Below is the class diagram showing methods, properties and enumerations available in the Publishing Module.



6.21.7.1 *Processing a configuration change notification*

The publishing processor component inherits from AbstractEDFSComponent which provides the stream line for the configuration change notification.

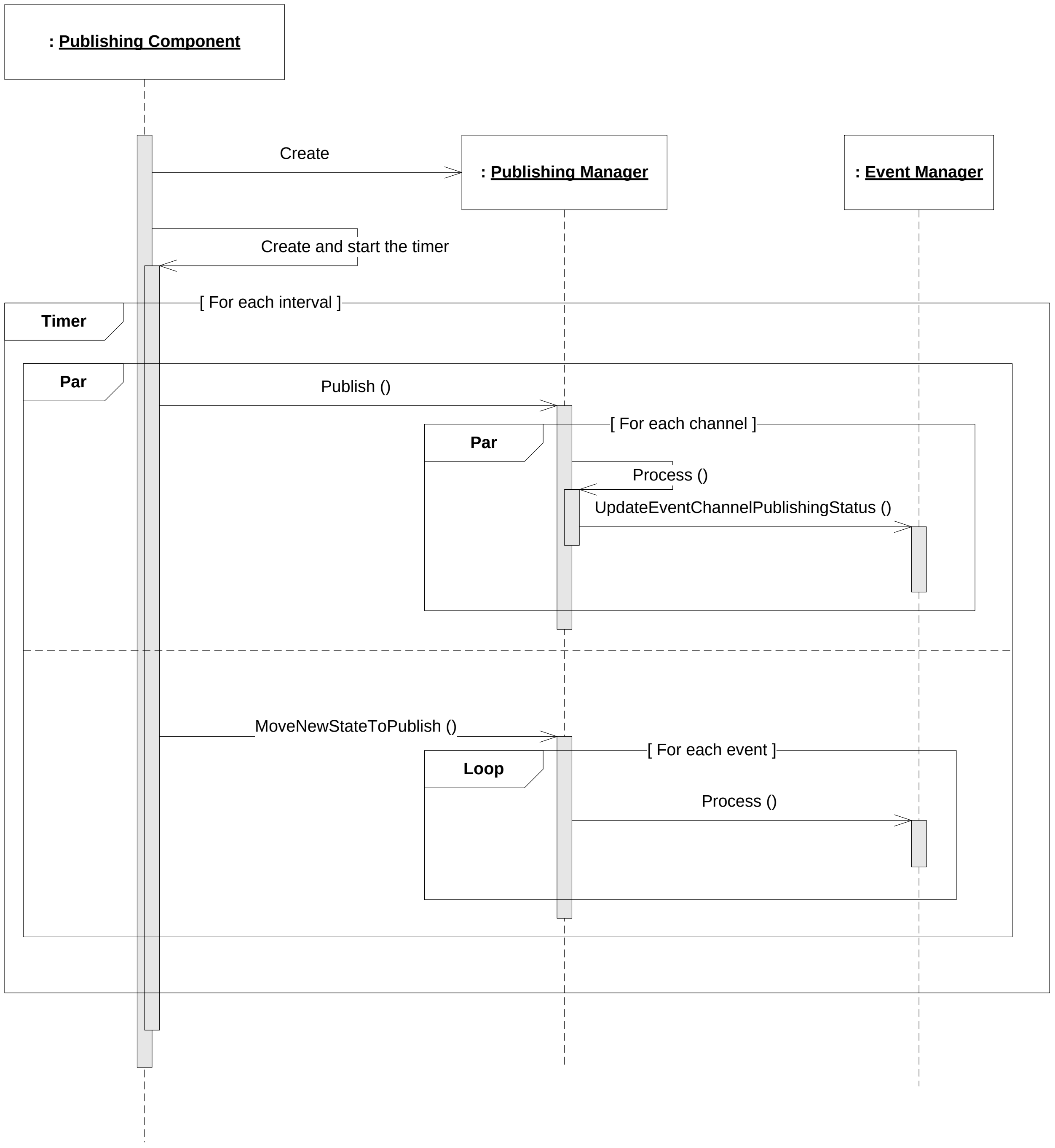
When the listener receives a change message, it would use cross-thread communication to notify the main thread to pause after processing the currently running threads which are created by the component, stop the timer, retrieve the updated configuration parameters, update the configurations, reset the timer and resume processing.

The following modules and classes are involved in configuration updates:

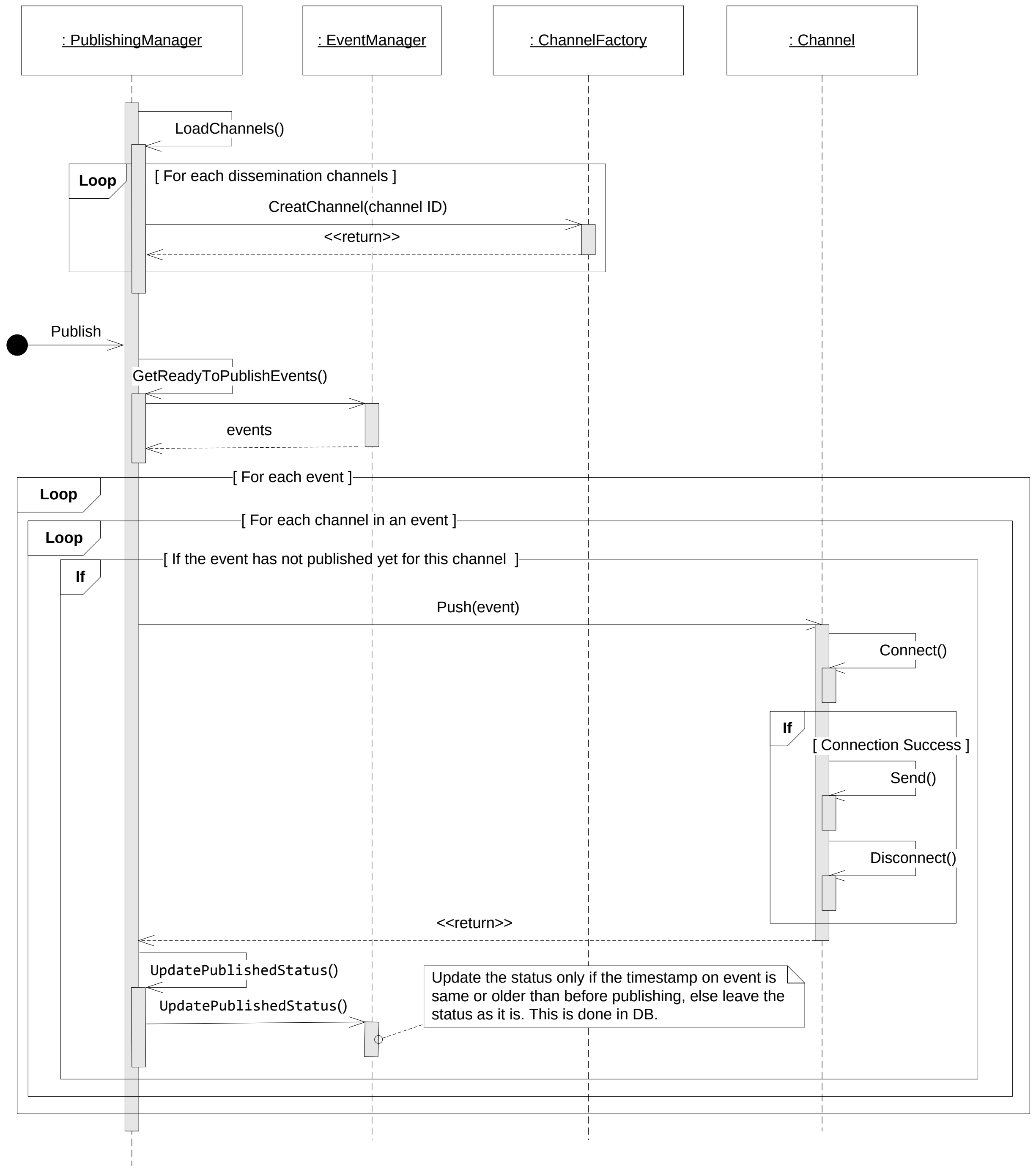
5. Alarm module,
6. Audit module,

7. Data module,
8. Event module
9. Log module,
10. Publishing module,
11. Queue manager,
12. User module and
13. Workflow module

6.21.7.2 *Example of publishing component process*



6.21.7.3 *Example of publishing the events*


: PublishingManager
: EventManager
: ChannelFactory
: Channel
LoadChannels()
Loop
[For each dissemination channels]
CreatChannel(channel ID)
<<return>>
Publish
GetReadyToPublishEvents()
events
Loop
[For each event]
Loop
[For each channel in an event]
If
[If the event has not published yet for this channel]
Push(event)
Connect()
If
[Connection Success]
Send()
Disconnect()
<<return>>
UpdatePublishedStatus()
UpdatePublishedStatus()
Update the status only if the timestamp on event is same or older than before publishing, else leave the status as it is. This is done in DB.

6.22 Reporting

This section describes each of the EDFS reports in detail.

6.22.1 Event Report

This report includes the Audit report as a sub report.

Fields	Columns to pull data from	Query
Event ID	ID column from tblEvents	GetEventReport(fromDate, toDate, myEventTypeID, myEventSubTypeID, myRoadName)
Event Description	Description column from tblEvents	Same
Road Name	Name column from tblFacilities where ID = tblEvents. RouteFacilityID	same

6.22.2 Event Audit Report

Fields	Columns to pull data from	Query
Date/Time Change	TimeStamp column from tblAuditRecords	GetEventAuditReport(myEventID)
Username	FirstName + LastName columns from tblUsers where ID = tblAuditRecords.UserID	Same
Field(s) Changed	FiledName column from tblAuditRecords	Same
Changed From	FromValue column from tblAuditRecords	Same
Changed To	ToValue column from tblAuditRecords	same

6.22.3 Event Activities Reports (per time period)

Fields	Columns to pull data from	Query

Average Event duration	Duration column from tbleventhistory where status = closed	GetEventActivitiesReports (fromDate, toDate, eventTypeID)
Number of Events collected	Distinct of tbleventhistory where status = new	same
Number of Events Modified	Distinct of tbleventhistory where status <> new	same
Number of Events Discarded	Distinct of tbleventhistory where status = discarded	Same
Number of Events Disseminated	Distinct of tbleventhistory where status = published	Same
Number of Event Closed	Distinct of tbleventhistory where status = closed.	same

6.22.4 User Performance Measures Reports (per time period)

Fields	**Columns to pull data from**	**Query**
Number of Published Events	Total count of ID from tbleventhistory where status = published per User	GetUserPerformanceMeasuresReports (fromDate, toDate, eventTypeID, eventSubTypeID, userID)
Average Event Handling Time	Distinct of tbleventedits (TBD) per User	TBD
Total Number of Events Modified	Total count of ID from tbleventhistory where status <> new per User	Same
Username	FirstName + LastName columns from tblUsers where ID = tblEventHistory.UserID	same

6.22.5 Current Performance and Activities Reports (at current date/time)

Fields	Columns to pull data from	Query
Number of logged in users	Total count of tblUserSessions	GetNumberOfCurrentLoggedInUsersReport ()
Number of Published Events	Total distinct count of tbleventhistory where status = published and event@current between StartDate and EndDate	GetNumberOfCurrentPublishedEventsReport (myEventTypeID, myEventSubTypeID)

6.23 Data Archiving

The data archiving will consists of archiving below list of tables:

- tblEvents, tblEventHistory, tblEventSchedules, tblEventScheduleHistory, tblAlarms, tblAlarmHistory, tblAuditRecords and tblLogMessages.

The data in the above tables are archived into the following tables:

- tblEvents_Archive, tblEventHistory_Archive, tblEventSchedules_Archive, tblEventScheduleHistory_Archive, tblAlarms_Archive, tblAlarmHistory_Archive, tblAuditRecords_Archive and tblLogMessages_Archive.

The schemas of the archive tables are similar to the original tables except that their primary key columns' "auto increment" feature is disabled and they have one additional column named "DateArchived". All necessary SQL statements and logic for data archiving are inside "ArchiveEvents" stored procedure. The events which are older than 15 days (which is a configurable) and are closed will be archived. After data has been successfully archived, the original data will be purged from respective tables.

This entire functionality is wrapped within ArchieveProcessor component. The archive processor component inherits from AbstractEDFSComponet.

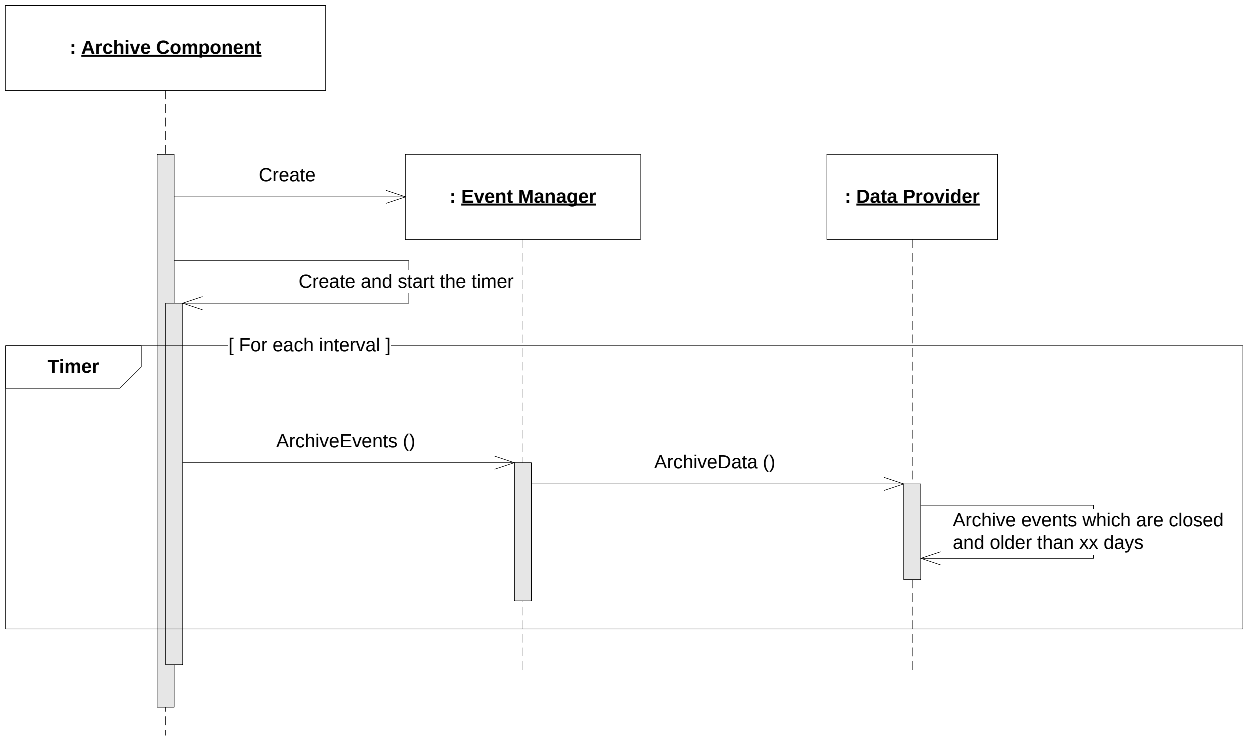
When the component receives a change message, it will use cross-thread communication to notify the main thread to pause after processing all the currently executing threads that were created by the component, stop the timer, retrieve the updated configuration parameters, update the configurations, reset the timer and resume processing.

The following modules and classes are involved in configuration updates:

1. Alarm module,
2. Audit module,
3. Data module,
4. Event module
5. Log module,

6. Queue manager and
7. User module

6.23.1 Example of archive component process



7 Technology Selection

7.1.1 Software and hardware requirements

The software and hardware requirement for new EDFS consists of:

- Relational Database – To fulfill all data storage needs per section 2.5 of Functional requirement document.
- Web Server – A server environment to host the EDFS website.
- Processing Server – A server environment that will run all the data interfaces, data processors, logging, and audit and event orchestration engine.
- Messaging Queue - To allow inter process message transfers between various components within EDFS.
- Web Framework – A software framework that will provide platform to build EDFS web application and services.
- Server Side Development Platform – A software programming language and runtime environment that will facilitate development of server side code.

- JavaScript framework - A JavaScript code library that provides inbuilt methods and utilities for faster and cleaner user interface code development.
- Mapping framework – A combination of server and development framework to create and consume maps within EDFS.

7.1.2 Key criteria

The key criteria for selecting any particular technology are:

- Prior experience – Was this particular technology successfully used in previous projects?
- Learning Curve – If there is no prior experience with this technology, how extensive is the learning curve?
- Integration - How well does this particular tool integrate with rest of the system, EDFS and other 511 applications?
- Hosting flexibility – How tightly coupled is this particular tool to hosting provider, in this case, AWS?
- Scalability Limitations– How well does this technology scale?
- Availability Limitations – Are there any limitations that would hinder developing a highly available solution?

Below tables lists how these technical requirements are met and justification for selection.

Requirement	**Selection**	**Prior experience**	**Learning Curve**	**Integration**	**Hosting Flexibility**	**Scalability Limitation**	**Availability Limitation**	**Notes**

Relational Database	MYSQL (Deployed as AWS RDS)	Yes	NA	**Low**	High	None	None	Keeping in line with MTC's future cloud based AWS hosting environment, MYSQL RDS has been selected. The RDS solution offers built-in replication for high availability without the additional burden. This also offers a very low risk in terms of moving away from AWS, incase MTC decides so. The current requirement doesn't entail deeper integration of database with other components besides being used as pure storage. Therefore the low level of integration is not perceived as a big risk to the project.
Web Server	IIS 7.5 running on windows 2008 in AWS.	Yes	NA	High	High	None	None	The web development framework will be ASP.NET 4.0, IIS 7.5 and higher is recommended server environment by Microsoft. Other existing MTC websites are using use IIS 7.5.
Processing Server	Windows 2008 Server	Yes	NA	High	High	None	None	Since the server side development framework will be C#, Windows server is the most appropriate choice.
Messaging Queue	MSMQ	Yes	NA	High	High	None	None	MSMQ provides a light-weight asynchronous message processing framework and integrates well with Microsoft .NET.
Web Framework	ASP.NET 4.0	Yes	NA	High	High	None	None	We have used this in new traffic website and all new projects are being developed using ASP.NET 4.0.
Server Side Development Platform	C# 4.0	Yes	NA	High	High	None	None	We have used this in new traffic website and all new

								projects are being developed using C# 4.0.
JavaScript framework	jQuery	Yes	NA	High	High	None	None	We have used this in new traffic website and many other projects.
Mapping framework	ArcGIS 10 and ArcGIS JavaScript	Yes	NA	High	High	None Non		We have used this in new traffic website. And the project is based on the assumption to reuse mapping capabilities form traffic website.

8 Windows Services

Multiple instances of a windows service such as the LogProcessor may be running at the same time, but only one will be active. All instances of a service will have a unique name and start in passive mode.

Each instance of the service will have a config file with similar entries:

[redacted]

Each passive instance will ping the service heartbeat at a unique (configurable) interval so that only one will try to become active eg:

- instance1 will check each minute at the 15 second mark
- instance2 will check each minute at the 30 second mark

When a passive instance detects that the service heartbeat is outdated it will:

- stop heartbeat ping
- update service activation history and heartbeat with its instance name and timestamp
- load component configuration settings from the ConfigDB
- start queue listener (and timer if it's a timed service)
- start timer to update the heartbeat

The outdate interval = 2 * number of instances * heartbeat interval

An active instance must be restarted to return to passive mode.

8.1 Service Installation

Each instance of the service must have its own folder with a copy of the exe, dll and config files eg:

- C:\Services\EDFSCore.LogProcessor\instances\one
- C:\Services\EDFSCore.LogProcessor\instances\two

The SC utility can be used to create, stop and delete the service instances:

8.1.1 Create the service instances

[redacted]

8.1.2 Delete the service instances

[redacted]

8.2 Service Monitoring

8.2.1 tblServiceActivationHistory

When an instance of a service becomes active, it updates the service activation history.

ServiceName	InstanceName	PreviousInstanceName	TimeSinceHeartbeat	Timestamp
SAIC EDFS LogProcessor	SAIC EDFS LogProcessor 1	SAIC EDFS LogProcessor 2	00:08:59	[redacted]

8.2.2 tblServiceHeartbeats

At every heartbeat interval, the active service updates the heartbeat.

ServiceName	InstanceName	Timestamp
SAIC EDFS LogProcessor	SAIC EDFS LogProcessor 1	[redacted]

8.3 Service Testing

The Test.EDFSCore.Interactive project contains a ServiceActivationTest form that can be used to start/stop and monitor instances of the LogProcessor service. When running in DEBUG mode, the services will write entries to the System Event log.

[redacted]

A session may look like this:

[redacted]